TUGAS AKHIR – KI141502

DESAIN DAN ANALISIS ALGORITMA CYCLE DETECTION IN WEIGHTED GRAPH PADA STUDI KASUS URI ONLINE JUDGE ELECTRICAL POLLUTION

MUHAMMAD FIRZA GUSTAMA
05111540000170

Dosen Pembimbing
Rully Soelaiman, S.Kom., M.Kom.
Dwi Sunaryono, S.Kom., M.Kom.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019

TUGAS AKHIR – KI141502

DESAIN DAN ANALISIS ALGORITMA CYCLE DETECTION IN WEIGHTED GRAPH PADA STUDI KASUS URI ONLINE JUDGE ELECTRICAL POLLUTION

MUHAMMAD FIRZA GUSTAMA
05111540000170

Dosen Pembimbing I
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing II
Dwi Sunaryono, S.Kom., M.Kom.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2019

[*Halaman ini sengaja dikosongkan*]

FINAL PROJECT - KI141502

DESIGN AND ANALYSIS OF ALGORITHM CYCLE DETECTION IN WEIGHTED GRAPH FOR SOLVING URI ONLINE JUDGE ELECTRICAL POLLUTION

MUHAMMAD FIRZA GUSTAMA
05111540000170

Supervisor I
Rully Soelaiman, S.Kom., M.Kom.

Supervisor II
Dwi Sunaryono, S.Kom., M.Kom.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
Sepuluh Nopember Institute of Technology
Surabaya, 2019

[*Halaman ini sengaja dikosongkan*]

LEMBAR PENGESAHAN

DESAIN DAN ANALISIS ALGORITMA CYCLE DETECTION IN WEIGHTED GRAPH PADA STUDI KASUS URI ONLINE JUDGE ELECTRICAL POLLUTION

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Mata Kuliah Algoritma Pemrograman
Program Studi S-1 Departemen Teknik Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:
Muhammad Firza Gustama
NRP: 051115 40000 170

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Rully Soelaiman, S.Kom., M.Kom.
NIP. 197002131994021001 ..... (Pembimbing 1)

Dwi Sunaryono, S.Kom., M.Kom.
NIP. 197205281997021001 ..... (Pembimbing 2)

SURABAYA
JANUARI 2019

[*Halaman ini sengaja dikosongkan*]

DESAIN DAN ANALISIS ALGORITMA CYCLE DETECTION IN WEIGHTED GRAPH PADA STUDI KASUS URI ONLINE JUDGE ELECTRICAL POLLUTION

Nama Mahasiswa : Muhammad Firza Gustama
NRP : 051115 40000 170
Departemen : Teknik Informatika FTIK - ITS
Dosen Pembimbing 1 : Rully Soelaiman, S.Kom., M.Kom.
Dosen Pembimbing 2 : Dwi Sunaryono, S.Kom, M.Kom.

Abstrak

Permasalahan dalam buku tugas akhir ini adalah permasalahan yang diambil dari dunia nyata namun sudah disederhanakan kedalam bentuk soal yang terdapat pada situs URI Online Judge "Electrical Pollution". Dalam permasalahan ini, diketahui bahwas generator yang digunakan untuk tenaga listrik rumah dan gedung menyebabkan anomali medan magnet lokal pada titik – titik tertentu pada suatu kota. Pada soal ini diberikan sebuah koordinat x dan y yang memiliki anomali. Suatu graf dibentuk dari setiap koordinat x dan y dengan bobot adalah anomali tersebut. Dicari anomali yang dihasilkan oleh generator pada koordinat lain.

Tugas akhir ini akan mengimplementasikan deteksi siklus ganjil pada graf menggunakan Breadth First Search dan struktur data graf bipartit dalam menyelesaikan permasalahan Electrical Pollution. Implementasi dalam tugas akhir ini menggunakan bahasa pemrograman C++. Hasil uji coba menunjukkan bahwa sistem mendapatkan anomali pada koordinat lain dengan benar dan memiliki pertumbuhan waktu dengan kompleksitas O(N + V + E) dimana V adalah jumlah vertex dan E adalah jumlah Edge.

Kata kunci: Breadth First Search, Graf Bipartit, Graf Siklus Ganjil

[*Halaman ini sengaja dikosongkan*]

DESIGN AND ANALYSIS OF ALGORITHM CYCLE DETECTION IN WEIGHTED GRAPH FOR SOLVING URI ONLINE JUDGE ELECTRICAL POLLUTION

Student Name	: Muhammad Firza Gustama
Registration Number	: 051115 40000 170
Department	: Informatics Department Faculty of IT – ITS
First Supervisor	: Rully Soelaiman, S.Kom., M.Kom.
Second Supervisor	: Dwi Sunaryono S.Kom, M.Kom.

Abstract

The problem in this final assignment book is taken from the real world problem but has been simplified into the form of the questions found on the URI Online Judge "Electrical Pollution" site. In this problem, it is known that generators used for home and building electric power cause anomalies of local magentic fields at certain points in a city. In this problem given an x and y coordinate which has an anomaly. A graph formed from each x and y coordinate with weight is the anomaly. Look for anomalies generated by the generator in other coordinates.

This final project will implement odd cycle detection in graphs using Breadth First Search and bipartite graph data structure in solving Electrical Pollution problems. Implementation in this final project uses C ++ programming language. The test results show that the system gets anomalies at other coordinates correctly and has a time growth with complexity O (N + V + E) where V is the number of vertices and E is the number of Edge.

Keywords: Biparttite Graph, Breadth First Search, Odd Cycle Graph

[*Halaman ini sengaja dikosongkan*]

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas pimpinan, penyertaan, dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul :

DESAIN DAN ANALISIS ALGORITMA CYCLE DETECTION IN WEIGHTED GRAPH PADA STUDI KASUS URI ONLINE JUDGE ELECTRICAL POLLUTION

Penelitian Tugas Akhir ini dilakukan untuk memenuhi salah satu syarat meraih gelar Sarjana di Departemen Teknik Informatika Fakultas Teknologi Informati Institut Teknologi Sepuluh Nopember.

Dengan selesainya Tugas Akhir ini diharapkan apa yang telah dikerjakan penulis dapat memeberikan manfaat bagi perkembangan ilmu pengetahuan terutama di bidang teknologi informasi serta bagi diri penulis sendiri selaku peneliti.

Penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan dukungan baik secara langsung maupun tidak langsung selama penulis mengerjakan Tugas Akhir maupun selama menempuh masa studi antara lain:

1. Terimakasih kepada Allah SWT masih diberi kesempatan, kesehatan dan umur untuk menempuh kuliah disini dan menjalani hidup dengan baik.
2. Bapak dan Ibu penulis yang selalu memberikan perhatian, dorongan, dan kasih sayang juga tunjangan makanan supaya lebih semangat menempuh kuliah maupun pengerjaan Tugas Akhir ini.
3. Bapak Rully Soelaiman, S.Kom., M.Kom. selaku Dosen Pembimbing yang telah membimbing saya selama masa kuliah maupun penyelesaian tugas akhir ini, memberi ilmu, nasihat, dan motivasi.
4. Bapak Dwi Sunaryono, S.Kom., M.Kom. Selaku dosen pembimbing yang telah memberikan ilmu, dan masukan kepada penulis.

5. Riris, Bram, sebagai orang yang selalu memperhatikan, menyemangati dan memotivasi saya untuk kuliah dan menyelesaikan tugas akhir ini.
6. ITS Jazz sebagai sumber penambah sangu saya dan refreshing saya.
7. Teman-teman angkatan 2015 jurusan Teknik Informatika ITS yang telah menemani perjuangan penulis selama 4 tahun masa perkuliahan.
8. Serta pihak-pihak lain yang tidak dapat disebutkan disini yang telah banyak membantu penulis dalam penyusunan Tugas Akhir ini.

Penulis mohon maaf apabila masih ada kekurangan pada Tugas Akhir ini. Penulis juga mengharapkan kritik dan saran yang membangun untuk pembelajaran dan perbaikan di kemudian hari. Semoga melalui Tugas Akhir ini penulis dapat memberikan kontribusi dan manfaat yang sebaik-baiknya.

Surabaya, Januari 2019

Muhammad Firza Gustama

DAFTAR ISI

DAFTAR GAMBAR

[*Halaman ini sengaja dikosongkan*]

DAFTAR TABEL

[*Halaman ini sengaja dikosongkan*]

DAFTAR KODE SUMBER

[*Halaman ini sengaja dikosongkan*]

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan masalah, batasan permasalahan, metodologi pembuatan Tugas Akhir, dan sistematika penulisan.

1.1 Latar Belakang

Perkembangan dunia teknologi informasi selama beberapa dekade terakhir sangatlah pesat. Dalam perkembangannya, teknologi informasi seringkali dijadikan solusi bagi permasalahan-permasalahan yang pernah ada yang sebelumnya diselesaikan secara manual oleh manusia.

Permasalahan *URI Online Judge Electrical Pollution* adalah sebuah masalah yang ada di dunia nyata namun telah disederhanakan kedalam bentuk soal. Generator yang digunakan untuk memberi tenaga listrik rumah atau gedung menyebabkan anomali medan magnet lokal pada koordinat tertentu. Tugas Akhir ini adalah untuk memprediksi anomali pada koordinat lain dari informasi – informasi yang sudah diberikan sebelumnya

Hasil dari Tugas Akhir ini diharapkan dapat mengimplementasikan solusi dari pengembangan permasalahan *Electrical Pollution* dengan pertanyaan seminimal mungkin sehingga diharapkan dapat memberikan kontribusi pada pengembangan ilmu pengetahuan dan teknologi informasi.

1.2 Rumusan Permasalahan

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah sebagai berikut:

1. Bagaimana menganalisis dan mendesain algoritma yang efisien dalam menyelesaikan permasalahan Electrical Pollution pada situs penilaian URI Online Judge?

2. Bagaimana mengimplementasikan algoritma yang sudah didesain untuk menyelesaikan permasalahan Electrical Pollution pada situs penilaian URI Online Judge?
3. Bagaimana menguji implementasi algoritma yang sudah dirancang untuk mengetahui kinerja dari implementasi yang telah dibuat?

1.3 Batasan Permasalahan

Permasalahan yang dibahas pada Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Implementasi algoritma menggunakan bahasa pemrograman C++.
2. Batas maksimum kasus uji adalah 10^4.
3. Batas koordinat X dan Y adalah $-10^7 \leq X, Y \leq 10^7$
4. Batas anomali A adalah $-10^4 \leq A \leq -10^4$.

1.4 Tujuan Pembuatan Tugas Akhir

Tujuan dari pembuatan Tugas Akhir ini adalah sebagai berikut:

1. Melakukan desain dan implementasi tentang permasalahan *Electrical Pollution* pada situs penilaian *URI Online Judge*
2. Menganalisis hasil penyelesaian permasalahan *Electrical Pollution* pada situs penilaian *URI Online Judge*
3. Melakukan uji coba untuk mengetahui kebenaran dan kinerja dari implementasi yang telah dilakukan

1.5 Manfaat Tugas Akhir

Tugas Akhir ini diharapkan dapat mengimplementasikan solusi dari permasalahan *Electrical Pollution* sehingga dapat memberikan kontribusi pada pengembangan ilmu pengetahuan dan teknologi informasi.

1.6 Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan Tugas Akhir ini yaitu:

1.6.1 Penyusunan Proposal Tugas Akhir

Tahap awal untuk memulai pengerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir. Pada proposal ini, penulis mengajukan gagasan untuk menyelesaikan permasalahan Electrical Pollution pada situs penilaian URI Online Judge.

1.6.2 Studi Literatur

Pada tahap ini dilakukan studi literatur yang membahas lebih dalam yang berkaitan dengan graf, deteksi siklus, Breadth First Search. Studi literatur didapatkan dari buku, internet, dan mater – materi kuliah yang berhubungan dengan metode yang akan digunakan.

1.6.3 Implementasi Perangkat Lunak

Implementasi merupakan tahap untuk membangun algoritma yang akan digunakan. Pada tahap ini dilakukan implementasi dari rancangan struktur data yang akan dimodelkan sesuai dengan permasalahan. Implementasi ini dikakukan dengan menggunakan bahasa pemrograman C++.

1.6.4 Pengujian dan Evaluasi

Pada tahap ini dilakukan dengan uji coba kebenaran dan kinerja solusi yang telah diimplementasi dengan melakukan pengiriman sumber kode sistem ke situs penilaian URI Online Judge pada permasalahan yang terkait dan melihat hasil umpan balik. Pengujian dilakukan dengan membandingkan kompleksitas hasil uji coba dengan kompleksitas hasil analisis.

1.6.5 Penyusunan Buku Tugas Akhir

Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi aplikasi perangkat lunak yang telah dibuat.

1.7 Sistematika Penulisan

Buku Tugas Akhir ini merupakan laporan secara lengkap mengenai Tugas Akhir yang telah dikerjakan baik dari sisi teori, rancangan, maupun implementasi sehingga memudahkan bagi pembaca dan juga pihak yang ingin mengembangkan lebih lanjut. Sistematika penulisan buku Tugas Akhir secara garis besar antara lain:

Bab I Pendahuluan

Bab ini berisi penjelasan latar belakang, rumusan masalah, batasan masalah dan tujuan pembuatan Tugas Akhir. Selain itu, metodologi pengerjaan dan sistematika penulisan laporan Tugas Akhir juga terdapat di dalamnya.

Bab II Dasar Teori

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan Tugas Akhir ini.

Bab III Analisis dan Perancangan Sistem

Bab ini berisi penjelasan tentang rancangan dari sistem yang akan dibangun.

Bab IV Implementasi

Bab ini berisi penjelasan implementasi dari rancangan yang telah dibuat pada bab sebelumnya. Implementasi disajikan dalam bentuk *pseudocode* disertai dengan penjelasannya.

Bab V Pengujian dan Evaluasi

Bab ini berisi penjelasan mengenai data hasil percobaan dan pembahasan mengenai hasil percobaan yang telah dilakukan.

Bab VI Kesimpulan dan Saran

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan perangkat lunak ke depannya.

[*Halaman ini sengaja dikosongkan*]

BAB II
DASAR TEORI

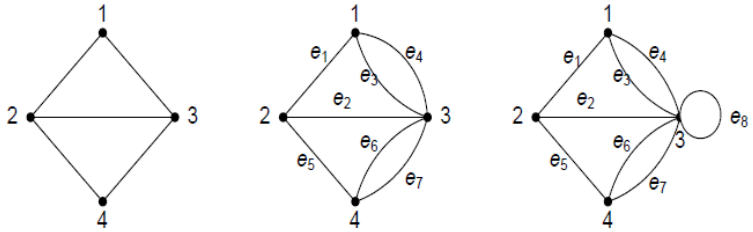
Pada bab ini akan dijelaskan mengenai dasar teori yang mejadi dasar pengerjaan Tugas Akhir ini.

2.1 Definisi Umum

Dalam subbab ini akan dibahas definisi-definisi umum yang akan digunakan dalam buku tugas akhir ini.

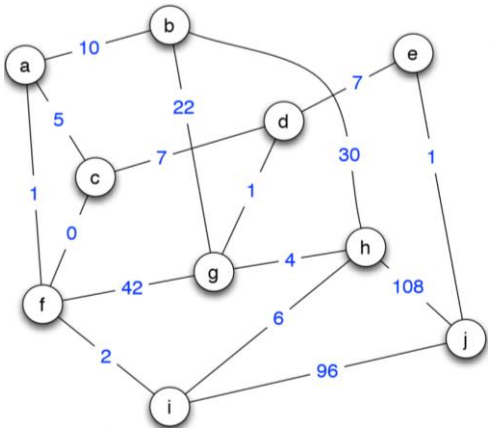
Suatu graf memiliki lebih dari satu titik (*vertex*) yang beberapa diantaranya dihubungkan oleh penghubung (*edge*). Secara formal, Sebuah graf *G* didefinisikan sebagai pasangan himpunan (*V, E*), dengan *V* adalah himpunan tak kosong dari simpul-simpul (*vertex*) pada *G*. Sedangkan *E* adalah himpunan rusuk (*edge*) pada *G* yang menhubungkan sepasang simpul. Himpunan simpul pada *G* dinotasikan sebagi *V*, dan himpunan rusuk pada *G* dinotasikan sebagai *E*, sehingga *G=(V, E)* [1].

Jika nilai dari *edge* (*u, v*) sama dengan *edge* (*v, u*), maka *edge* tersebut berarti tak-berarah, atau disebut juga dengan *undirected edge*, begitupun sebaliknya, apabila tidak sama maka disebut sebagai *directed edge* atau *edge* berarah. Suatu graf pasti berisi *edge* yang sejenis, yaitu berarah atau tak-berarah saja. Graf yang terdiri dari *directed edge*, disebut dengan *directed graph* atau graf berarah. Sedangkan graf yang terdiri dari *undirected edge*, disebut dengan *undirected graph* atau graf tak-berarah [2].



Gambar 2.1.1 Graf Tidak Berarah

Suatu graf terkadang memiliki beberapa varian yang khusus, seperti adanya suatu *integer* pada *edge*. Bilangan *integer* tersebut dapat menjadi informasi tambahan antara suatu *vertex* dengan *vertex* lainya sebagai contoh jarak, waktu, dsb. Varian *integer* pada *edge* tersebut adalah *weight*, Graf yang memiliki *edge* dengan *weight* disebut *weighted graph* atau graf berbobot. Jika $G = (V, E)$ adalah graf maka bobot W adalah $W : E \rightarrow R^+$ [5].

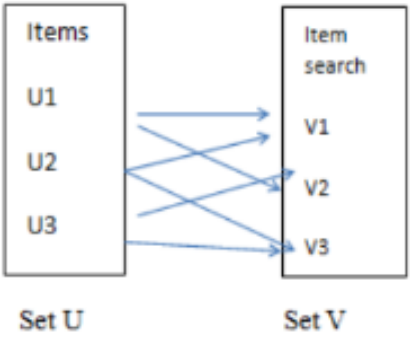


Gambar 2.1.2 Graf Berbobot

Salah satu permasalahan yang umum pada graf adalah mencari siklus (*cycle*) atau sirkuit (*circuit*) permaslahan tersebut adalah *cycle finding*. Siklus pada graf dapat dibagi menjadi dua yaitu *odd cycle* atau siklus ganjil dan *even cycle* atau siklus genap. Pada permasalahan tugas akhir ini akan lebih fokus untuk mencari siklus ganjil pada graf terkait penyelesaian persoalan tugas akhir ini. Graf yang memiliki siklus atau sirkuit adalah graf dengan sebuah lintasan *edge* berawal dan berakhir pada *vertex* yang sama, Graf tersebut adalah *Cycle Graph* atau Graf Siklus, Secara formal *Cycle Graph* atau Graf Siklus siklus adalah graf dengan *n vertex* yang memiliki satu siklus yang melewati semua *vertex* [7]. Siklus

terjadi hanya pada graf dengan jumlah *vertex* 3 atau lebih. Contoh siklus dari graf pada Gambar 2.1.2 adalah a – c – f – a.

Metode sederhana tentang penyelesaian terkait permasalahan *cycle finding* adalah dengan menggunakan bentuk lain pada graf atau menggunakan struktur data lain pada graf tersebut. Struktur data lain tersebut adalah dengan menggunakan dua struktur data *disjoint sets*. *Sets* adalah himpunan dari beberapa data yang unik [5], maka *disjoint sets* adalah dua atau lebih *sets* dengan data yang unik dan berbeda antara satu *sets* dengan *sets* lainya. Graf yang menggunakan struktur data *disjoint sets* adalah *Bipartite Graph* atau graf bipartit. Secara formal, Graf bipartit adalah graf dengan *vertex* yang dapat dibagi menjadi dua *disjoint sets* U dan V (dimana U dan V berada pada *sets* berbeda lalu setiap *edge* terhubung dengan satu *vertex* pada U dan satu *vertex* pada V). *Vertex* pada U dan V sering disebut *partite sets*. Secara ekuivalen, graf bipartit adalah graf yang tidak memiliki siklus ganjil [6]. Siklus ganjil pada graf bipartit dapat ditemukan jika dan hanya jika suatu elemen terhubung dengan elemen lain pada satu *sets* yang sama. Jadi untuk menemukan bahwa graf tersebut memiliki siklus ganjil atau tidak adalah dengan menentukan bahwa graf tersebut adalah graf bipartit atau bukan. Jika suatu elemen pada *sets* yang sama saling berhubungan maka graf tersebut bukan graf bipartit.



Gambar 2.1.3 Graf Bipartit

2.2 Algoritma *Cycle Finding*

Permasalahan pada *cycle finding* untuk menentukan bahwa graf tersebut memiliki *odd cycle* atau tidak dapat diselesaikan secara sederhana dengan menentukan apakah graf tersebut graf bipartit atau bukan. Jika graf tersebut adalah graf bipartit maka graf tersebut tidak akan memiliki *odd cycle* begitu juga sebaliknya jika graf tersebut adalah bukan graf bipartit maka graf tersebut tidak memiliki *odd cycle*.

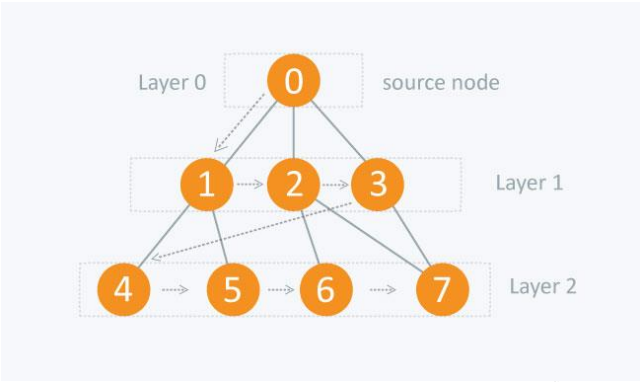
Berikut merupakan langkah – langkah dari algoritma untuk menentukan graf bipartit atau bukan.

1. Letakan *source vertex* pada *sets U*
2. Letakan *neighbor vertex* pada *sets V*
3. Letakan *neighbor's neighbor vertex* pada *sets U*
4. Jika ditemukan *neighbor* berada *sets* yang sama dengan *vertex* sekarang, maka tandai bahwa graf bukan bipartit dan dapatkan *w*(*root*) dan lanjutkan hingga semua *vertex* dikunjungi
5. Setelah iterasi, Jika graf tidak bipartit maka ubah tiap *weight vertex*

Untuk menentukan *neighbor vertex* dari *source vertex* diperlukan *graph traversal*, ada banyak metode *traversal* pada graf namun untuk memntukan semua *neighbor vertex* dari *source vertex* diperlukan *traversal Breadth First Search*. *Breadth First Search* (BFS) adalah salah satu cara untuk melakukan graf traversal. BFS dimulai dari memilih salah satu *vertex* dan mentraversal layer per layer sehingga mengeksplor *neighbor vertex* yang terhubung dengan *source vertex* [8].

Traversal pada BFS dapat disimpulkan seperti langkah – langkah berikut:

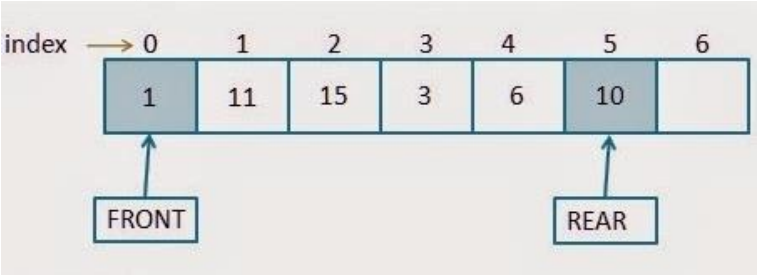
1. Dari *source vertex* temukan semua *neighbor vertex*
2. Kunjungi layer selanjutnya yaitu layer *neighbor vertex*
3. Ulangi langkah 1 – 2.



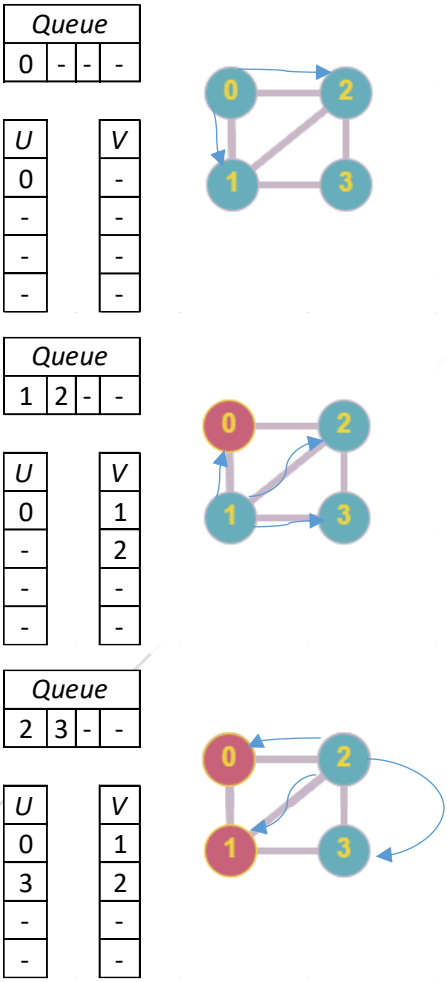
Gambar 2.2.1 *Breadth First Search Traversal*

Pada *traversal* BFS memerlukan struktur data *queue*. Struktur data *Queue* berperan penting dalam algoritma BFS. Dalam artian *queue* berarti antrian dan memiliki konsep FIFO (*first in first out*). Dalam penggunaan *queue* dengan BFS dapat disimpulkan seperti langkah – langkah berikut:

1. Pilih *source vertex* dan masukan kedalam *queue*
2. Masukan tiap *neightbor vertex* ke dalam *queue*
3. *Vertex* yang dikunjungi keluar dari *queue* lalu lanjutkan ke *vertex* selanjutnya yang berada pada *queue*
4. Kembali ke step 2 hingga semua *vertex* sudah dikunjungi



Gambar 2.2.2 Queue



Gambar 2.2.3 Ilustrasi Algoritma *Cycle Finding*

Dapat dilihat pada Gambar 2.2.3 ketika *source vertex* adalah *vertex* 1 maka *neighbor vertex* adalah 0, 2, 3. Lalu diketahui bahwa

vertex 1 dan *vertex* 2 berada pada *sets* yang sama maka graf tersebut bukanlah graf bipartit.

2.3 Struktur Data *Pair*

Struktur data *Pair* digunakan pada tahap desain dan implementasi untuk mempermudah *code* program dan lebih efisien dalam hal memori dan waktu program. Struktur data ini tersedia dalam *library* C++. Secara formal, struktur data *Pair* adalah sebuah struktur data yang dapat menyimpan dua objek menjadi sebuah kesatuan. [8]

2.4 Struktur Data *Map*

Struktur data *Map* juga digunakan pada tahap desain dan implementasi untuk mempermudah *code* program dan lebih efisian dalam hal memori dan waktu program. Struktur data ini tersedia dalam *libarry* C++. Secara formal, *Map* adalah struktur data yang menyimpan elemen dengan cara dipetakan. Setiap elemen memiliki kunci indeks dan data itu sendiri. Tidak ada elemen yang memiliki kunci indeks yang sama. [9]

2.5 Deskripsi Umum Permasalahan

Permasalahan yang diangkat dalam tugas akhir ini diangkat dari suatu permasalahan yang terdapat pada situs penilaian *URI Online Judge* yaitu *Electrical Pollution* dengan kode soal 1334 [1], deskripsi soal dari sumber asli menggunakan bahasa Inggris terdapat pada Gambar 2.5.1.

Pada permasalahan *Electrical Pollution* diceritakan mengenai sebuah kota fiktif bernama *Sortonia* ibu kota provinsi *Nlogonia* utara. Kota tersebut memiliki struktur yang unik yaitu kotak – kotak yang terbentang dari utara, selatan, timur dan barat. Namun terdapat *Merge Avenue* yang yang berada pada arah diagonal kota tersebut.

Universitas lokal pada kota tersebut sedang melakukan pengembangan generator tenaga magnet yang berletak di *Merge Avenue* yaitu pada sisi diagonal kota tersebut untuk memberi tenaga listrik untuk semua rumah dan bisnis kota tersebut.

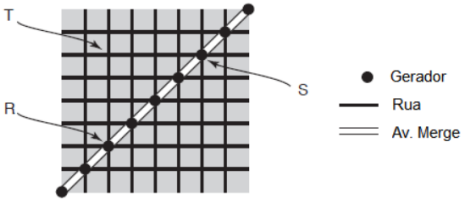
Sortonia is the capital of the North Nlogonia province. The city is laid out with almost all of its streets in a square grid, aligned to either the North-South or the West-East direction. The only exception is Merge Avenue, which runs Southwest-Northeast, splitting city blocks along their diagonals.

Sortonia is also one of the greenest cities in Nlogonia. The local university developed technology to harness the magnetic field of Earth for energy generation. As a consequence, all intersections of Merge Avenue have power generators installed, supplying all the homes and businesses of the city.

This technology was praised by environmentalists at the time for eliminating Sortonia's carbon footprint, but soon after its introduction, thousands of bees and birds were found dead in the city. Puzzled, the Queen of Nlogonia ordered the queendom's biophysicists to investigate the phenomenon.

After many months of study, they discovered that the generators used by Sortonians created anomalies in the local magnetic field. The birds and bees that use the Earth's magnetic field to guide their flight were confused by these anomalies, started flying in circles and eventually died of exhaustion.

According to the biophysicists' theoretical models, each generator creates an anomaly that is represented as an integer value. Each anomaly propagates indefinitely in all four compass directions. Points that are not directly north, south, west or east of the generator are unaffected by it. On the other hand, if a point is aligned with two generators then the anomaly at that point is the sum of the two anomalies produced by those generators. As an example, consider the picture below that represents a certain portion of Sortonia. The anomaly at point R is just the one produced by the generator at that point, while the anomaly at point T is the sum of the anomalies produced by the generator at point R and the generator at point S.



The biophysicists would like to measure the anomalies for some city intersections, but these measurements require expensive equipment and technical expertise. So they plan to measure only a subset of the city's intersections and extrapolate other data from them. Predicting an anomaly from a set of measurements might require combining several of them in complicated ways. Thus, the Queen ordered you to write a program that predicts the anomalies at certain intersections, given the measurements previously made.

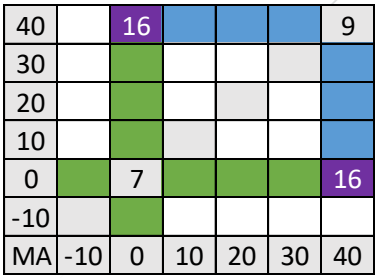
Gambar 2.5.1 Deskripsi Permasalahan Electrical Pollution pada URI Online Judge

Teknologi yang dikembangkan tersebut sangat di apresiasi oleh para pecinta lingkungan karena telah menghilangkan jejak karbon kota tersebut, namun setelah beberapa saat teknologi tersebut berjalan banyak sekali lebah, burung mati di kota tersebut. Tidak tahu kenapa ratu negri tersebut langsung pesuruh kerajaan untuk menginvestigasi fenomena tersebut.

Setelah berbulan – bulan mempelajari dan menginvestigasi fenomena tersebut, mereka menemukan bahwa generator yang digunakan kota *Sortonia* menyebabkan anomali pada medan magnet local. Burung dan lebah yang menggunakan medan magnet

bumi untuk terbang kebingungan mencari jalan sehingga banyak yang mati karena terbang terus hingga kelelahan.

Biophysicist memerlukan alat yang banyak dan mahal untuk mengukur anomali medan magnet lokal pada kota tersebut maka mereka akan hanya mengukur beberapa titik anomali pada kota tersebut lalu memprediksi sisanya. Lalu ratu kota *Sortonia* menyuruh untuk membuat program dengan memberi beberapa koordinat anomali dan besar anomali tersebut lalu prediksi sisanya. Diketahui bahwa generator hanya menyebabkan anomali pada arah utara, selatan, timur, dan barat lalu pada koordinat tertentu anomali tersebut adalah jumlah dari anomali pada generator. Ilustrasi anomali pada medan magnet lokal yang disebabkan generator dapat dilihat pada Gambar 2.5.2.



Gambar 2.5.2 Ilustrasi Anomali Medan Magnet yang disebabkan oleh Generator

Diketahui bahwa generator yang berada pada kota tersebut berletak pada sisi diagonal kota. Pada contoh Gambar 2.5.2 anomali pada medan magnet lokal yang diketahui adalah generator pada titik (0, 0) dengan anomali adalah 7 dan titik (40, 40) dengan anomali adalah 9. Sehingga anomali yang disebabkan pada dua titik tersebut terletak pada koordinat (40, 0) dan (0, 40) dengan anomali adalah jumlah dari anomali pada kedua generator tersebut yaitu 7 + 9 = 16.

Sebagai contoh dengan soal seperti berikut, diketahui bahwa koordinat x, y yaitu (30, -10) memiliki anomali sebesar 3, (30, 20)

memiliki anomali sebesar 15 dan (40, 20) memiliki anomali sebesar 2. Prediksilah anomali pada titik (-10, 40), (40, -10), (-10, -10). Ilustrasi untuk contoh soal terdapat pada Gambar 2.5.3 dimana MA adalah *Merge Avenue*.

40	?					
30						
20					15	2
10						
0						
-10	?				3	?
MA	-10	0	10	20	30	40

Gambar 2.5.3 Ilustrasi Contoh Soal

Dapat dilihat pada Gambar 2.5.3 untuk menemukan anomali pada koordinat (40, -10) diperlukan untuk mengetahui anomali generator pada *Merge Avenue* (-10, -10) dan (40, 40). Cara penyelesaian sederhana dapat dilihat pada Gambar 2.5.4 dengan menggunakan 3 koordinat anomali yang telah diketahui dapat ditemukan anomali lain sehingga dapat disimpulkan bahwa semakin banyak data yang diketahui semakin banyak pula prediksi anomali yang dapat dilakukan. Maka untuk koordinat (-10, 40) anomalinya adalah -10, (40, -10) anomalinya adalah -10 dan (-10, -10) tidak diketahui anomalinya.

Namun persoalan ini dapat diselesaikan lebih mudah dan sederhana lagi ketika data yang diberikan mampu memprediksi anomali pada generator di *Merge Avenue*. Karena dengan hasil prediksi tersebut dapat dengan secara langsung memprediksi dengan menjumlahkan anomali pada generator dengan generator lainya atau menggunakan sifat substitusi untuk mendapatkan anomali pada koordinat generator lain, seperti contoh pada Gambar 2.5.5 dapat dilihat bahwa hasil prediksi adalah hasil dari sifat substitusi matematika. Dari hasil prediksi tersebut maka dapat diprediksi juga dengan mudah anomali pada titik (-10, 40) dan (40,

-10) adalah 6, anomali pada titik (0, 40) dan (40, 0) adalah 7, anomali pada titik (10, 40) dan (40, 10) adalah 8.

40						d
30					c	
20				b	15	2
10						
0						
-10	a				3	-10
MA	-10	0	10	20	30	40

a + c = 3
b + c = 15
b + d = 2
a + d = (a + c) - (b + c) + (b + d)
a + d = 3 - 15 + 2 = -10

Gambar 2.5.4 Ilustrasi Penyelesaian Contoh Soal

Pada permasalahan Electrical Pollution diberikan koordinat *X*, *Y* dan anomali *A* sebanyak *M*. *A* adalah jumlah anomali pada generator yang berada pada *X* dan *Y*. Dengan koordinat dan anomali tersebut hitung anomali pada koordinat lain yang bersangkutan. Lalu diberikan koordinat *X*, *Y* sebanyak *Q* untuk mencari *A* pada koordinat tersebut.

Format masukan pada baris pertama diberikan bilangan bulat *M* dan *Q* yang merupakan jumlah koordinat anomali yang diketahui dan jumlah koordinat anomali yang akan dicari. Pada baris kedua terdapat M bilangan bulat yang merupakan *X*, *Y*, dan *A* merepresentasikan koordinat anomali yang diketahui. Pada baris selanjutnya terdapat *Q* bilangan bulat *X*, *Y* yang merupakan koordinat anomali yang akan dicari.

40						5
30						
20						
10			c			
0		b	5			
-10	a	3	4			
MA	-10	0	10	20	30	40

a + b = 3
a + c = 4
b + c = 5
a = ((a + b) + (a + c) - (b + c)) / 2 = (3 + 4 - 5) / 2 = 1
a + b = 3 -> 1 + b = 3 -> b = 3 - 1 = 2
a + c = 4 -> 1 + c = 4 -> c = 4 - 1 = 3

Gambar 2.5.5 Ilustrasi Penyelesaian Contoh Soal 2

Tiap input dari *Q* akan ada keluaran yang terdiri dari satu baris yaitu '*' jika informasi yang diberikan tidak cukup untuk memprediksi anomali pada koordinat tersebut, menampilkan sebuah bilangan integer jika informasi yang diberikan cukup untuk memprediksi anomali pada koordinat tersebut. Deskripsi format masukan dapat dilihat pada Gambar 2.5.6 dan contoh masukan pada Gambar 2.5.7 lalu deskripsi format keluaran dapat dilihat pada Gambar 2.5.8 dan contoh keluaran pada gambar pada Gambar 2.5.9.

Batasan dari permasalahan *Electrical Poluution* adalah sebagai berikut:

1. $1 \leq M, Q \leq 10^4$
2. $-10^7 \leq X, Y \leq 10^7$
3. $-10^4 \leq A \leq 10^4$
4. Batas Waktu: 3 detik

Input

Each test case is described using several lines. The first line contains two integers **M** and **Q** representing respectively the number of measurements and the number of queries ($1 \le M, Q \le 10^4$). Each of the next **M** lines describes a measurement using three integers **X, Y** and **A**, indicating that the measured anomaly at point **(X, Y)** is **A** ($-10^7 \le X, Y \le 10^7$ and $-10^4 \le A \le 10^4$). After that, each of the next **Q** lines describes a query using two integers **X'** and **Y'**, indicating that the anomaly at point **(X', Y')** must be predicted ($-10^7 \le X', Y' \le 10^7$). All positions are measured in city blocks; the first coordinate increases from West to East, while the second coordinate increases from South to North. Point (0, 0) is located on Merge Avenue. You may assume that within each test case each point is not measured more than once. Likewise, each point is not queried more than once. You may also assume that all the measurements are consistent.

The last test case is followed by a line containing two zeros.

Gambar 2.5.6 Deskripsi Masukan Soal

Sample Input

```
3 3
30 -10 3
30 20 15
40 20 2
-10 40
40 -10
-10 -10
6 8
0 1 11
0 3 8
1 0 11
3 0 8
4 4 0
3 5 6
1 5
0 3
3 0
4 3
0 2
2 4
4 4
5 5
0 0
```

Gambar 2.5.7 Contoh Masukan Soal

Output

For each test case output **Q + 1** lines. In the i-th line write the answer to the i-th query. If the information given by the measurements is enough to predict the anomaly at the queried point, then write an integer representing the predicted anomaly at the queried point. Otherwise write the character '*' (asterisk). Print a line containing a single character '-' (hyphen) after each test case.

Gambar 2.5.8 Deskripsi Keluaran Soal

Sample Output

```
-10
-10
*
-
9
8
8
*
*
*
0
*
-
```

Gambar 2.5.9 Contoh Keluaran Soal

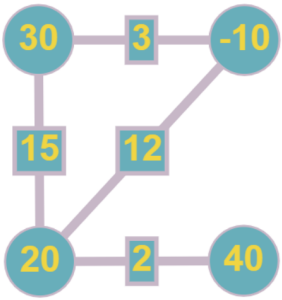
2.6 Penyelesaian Permasalahan *Electrical Pollution*

Permasalahan *Electrical Pollution* dapat diselesaikan dengan merubah representasi anomali pada koordinat tertentu yang dihasilkan oleh generator menjadi graf, lalu menggunakan algoritma *cycle finding*.

40						d
30					c	
20				b	15	2
10						
0						
-10	a			12	3	
MA	-10	0	10	20	30	40

Gambar 2.6.1 Representasi Soal

Dalam permasalahan ini representasi soal yang didapatkan merupakan *Adjacency Matrix* seperti pada Gambar 2.6.1, lalu cara pertama untuk menyelesaikan permasalahan tersebut adalah dengan merubah representasi menjadi graf. Cara mengubah representasi soal adalah dengan mengubah tiap koordinat (*x, y*) menjadi *vertex* lalu anomali dari generator adalah *weight* pada *edge* graf tersebut. Sebagai contoh dapat dilihat pada Gambar 2.6.2.

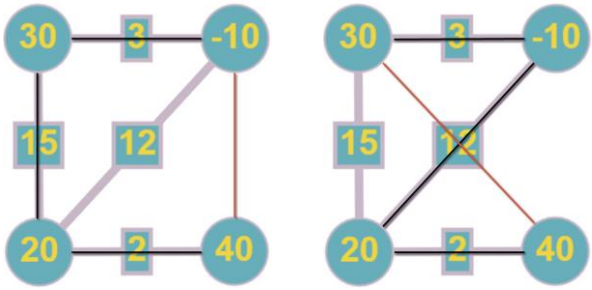


Gambar 2.6.2 Representasi Graf pada Permasalahan *Electrical Pollution*

Dari contoh Gambar 2.6.2 representasi graf yang dihasilkan berdasarkan pada permasalahan berikut. Diketahui bahwa anomali pada koordinat (30, -10) atau (-10, 30) adalah 3, anomali pada koordinat (30, 20) atau (20, 30) adalah 15, anomali pada koordinat (40, 20) atau (20, 40) adalah 2, dan anomali pada koordinat (-10, 20) atau (20, -10) adalah 12. Dapat dilihat bahwa titik *x* dan titik *y* tidak berpengaruh jika ditukar karena pada deskripsi soal tiap generator akan memberi anomali ke arah utara, selatan. barat, dan timur sehingga lebih memudahkan untuk penyelesaian soal *Electrical Pollution* sehingga memori dan waktu lebih efisien. Penentuan bahwa graf tersebut adalah graf bipartit atau tidak akan dijelaskan pada metode penggunaan algoritma *cycle finding* subbab 2.6.2.

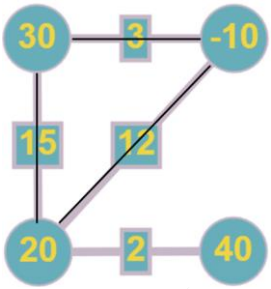
Penyelesaian permasalahan *Electrical Pollution* adalah dengan melakukan prediksi melalui data yang telah diberikan, maka prediksi semua kemungkinan pada Gambar 2.6.1 adalah sebagai berikut.

$a + d = (a + c) + (b + d) - (b + c) = 3 + 2 - 15 = -10$
$c + d = (a + c) + (b + d) - (a + b) = 3 + 2 - 12 = -7$



Gambar 2.6.3 Prediksi Semua Kemungkinan 1

a = ((a+b) + (a+c) - (b+c)) / 2 = (12 + 3 - 15) / 2 = 0
a + b = 0 + b = 12, maka b = 12
a + c = 0 + c = 3, maka c = 3
b + d = 12 + d = 2, maka d = -10
sehingga prediksi tiap pasangan *vertex* dapat dilakukan
a + d = 0 + -10 = -10
c + d = 3 + -10 = -7



Gambar 2.6.4 Prediksi Semua Kemungkinan 2

Dapat dilihat pada Gambar 2.6.3 dan Gambar 2.6.4 bahwa semua prediksi dari data yang diberikan hanya dapat dilakukan jika dan hanya jika *vertex* membentuk *odd cycle* dan *vertex* akan mementuk *even cycle* dengan satu *edge* yang hilang adalah hasil prediksi-nya. Maka diperlukan algoritma *cycle finding* untuk menyelesaikan permasalahan tersebut.

Dengan merubah graf pada Gambar 2.6.2 menjadi graf bipartit sekaligus menentukan apakah graf tersebut bipartit atau bukan kita dapat dengan mudah menghitung prediksi yang dihasilkan dari koordinat anomali yang dihasilkan, selain itu kompleksitas dan *run time* program akan lebih singkat dan efisien.

2.6.1 Penyelesaian Naif

Untuk menyelesaikan permasalahan ini dapat dilihat bahwa prediksi pada titik *x* dan *y* dapat dilakukan apabila representasi graf dengan *vertex x* dan *vertex y* membentuk *odd cycle* atau akan membentuk *even cycle*. Dengan kata lain bahwa prediksi dapat dilakukan bila *vertex x* dan *vertex y* saling berhubungan, maka permasalahan dapat diselesaikan dengan menemukan jalur antara *vertex x* dan *vertex y*.

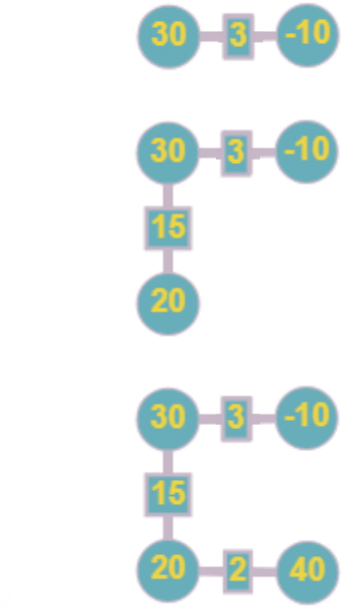
Pada masukan awal graf harus memiliki kriteria yaitu tidak memiliki *cycle*. Namun ketika pada masukan membentuk *cycle* yaitu dimana *vertex* dengan *vertex* lainya yang sudah memiliki *edge* saling terhubung maka cari *path* dua *vertex* tersebut dan lakukan perhitungan untuk memprediksi anomali pada titik tersebut yaitu titik generator. Contoh pembentukan graf dapat dilihat pada Gambar 2.6.1.1.

Pada masukan terakhir yaitu 20 -10 2 diketahui bahwa masukan tersebut membentuk *cycle* pada graf maka sistem akan melakukan pencarian *path* dari *vertex* 20 ke *vertex* -10 dengan step pada Gambar 2.6.1.2 maka *path* yang dihasilkan adalah -10 – 30 – 20 sehingga dengan *path* tersebut akan diprediksi anomali nya dengan persamaan berikut.

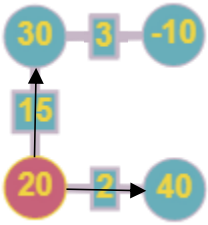
$$w(30, 30) = (w(-10, 30) + w(30, 20) – w(-10, 20)) / 2$$
$$w(30, 30) = (3 + 15 – 12) / 2$$
$$w(30, 30) = 3$$

Karena anomali yang didapatkan adalah anomali pada titik generator ($x = y$) yaitu pada titik (30, 30) maka lakukan iterasi tiap *vertex* yang berhubungan dengan *vertex* tersebut dan cari anomali pada titik generator lainya. Jika ada masukan pada titik (x, y) dimana anomali(x, x) atau anomali(y, y) diketahui maka didapatkan juga anomali pada titik lainya (jika anomali(x, x) diketahui maka hitung anomali(y, y) dan sebaliknya), prediksi ini dilakukan jika *vertex* membentuk *odd cycle*. Lalu bagaimana prediksi pada *even cycle*?

Contoh masukan		
30	-10	3
30	20	15
20	40	2
20	-10	12



Gambar 2.6.1.1 Pembentukan Graf (*odd cycle*)

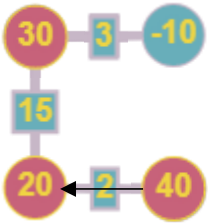


queue	20

vertex	*source*
20	-

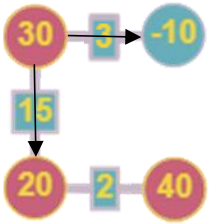
queue	40	30

vertex	*source*
20	-
30	20
40	20



queue	30

vertex	*source*
20	-
30	20
40	20



queue	-

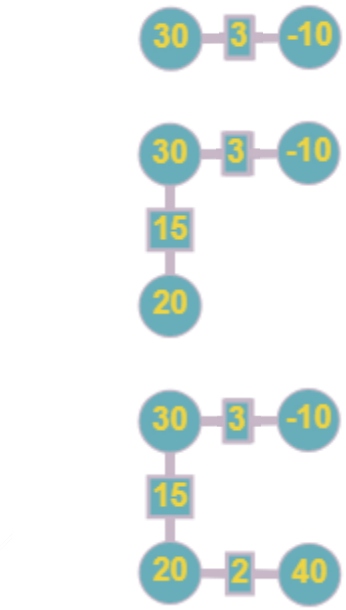
vertex	*source*
20	-
30	20
40	20
-10	30

Gambar 2.6.1.2 Pencarian *Path (odd cycle)*

Prediksi *even cycle* dapat dilakukan dengan sederhana yaitu dengan mencari *path* antara *source* dan *destination*. Jika jumlah *vertex* pada *path* berjumlah ganjil maka prediksi tidak dapat dilakukan namun jika jumlah *vertex* pada *path* berjumlah genap maka prediksi dapat dilakukan. Sebagai contoh masukan pada

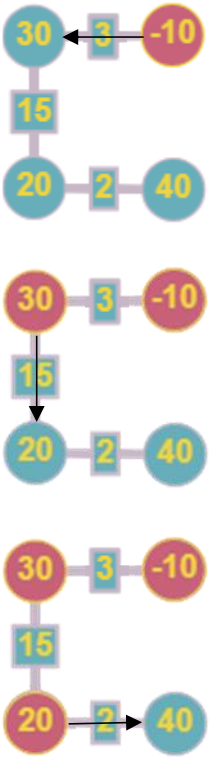
Gambar 2.6.1.3 dengan prediksi yang dilakukan adalah prediksi anomali pada titik -10 dan 40.

Contoh masukan		
30	-10	3
30	20	15
20	40	2



Gambar 2.6.1.3 Pembentukan Graf (*even cycle*)

Lalu untuk prediksi akan dilakukan pencarian *path* antara *vertex* -10 dan *vertex* 40 dengan ilustrasi pada Gambar 2.6.1.4 maka ditemukan *path* dengan rute 40 – 20 – 30 – -10 .



queue	-10

vertex	*source*
-10	-

queue	30

vertex	*source*
-10	-
30	-10

queue	20

vertex	*source*
-10	-
30	-10
20	30

queue	-

vertex	*source*
-10	-
30	-10
20	30
40	20

Gambar 2.6.1.4 Pencarian *Path* (*even cycle*)

Setelah pencarian *path* ditemukan akan dilakukan prediksi anomali menggunakan *path* yang didapat dengan persamaan seperti berikut

$w(-10, 40) = w(40, 20) – w(20, 30) + w(30, -10)$

$w(-10, 40) = 2 – 15 + 3 = -10$

Maka anomali pada titik (-10, 40) adalah -10.

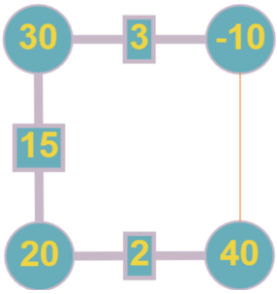
2.6.2 Penyelesaian dengan Algoritma *Cycle Finding*

Dalam permasalahan ini, algoritma *cycle finding* digunakan untuk menyelesaikan permasalahan tersebut. Diketahui bahwa prediksi dapat dilakukan jika dan hanya jika *vertex* membentuk *odd cycle* dan *vertex* akan membentuk *even cycle*, namun jika graf telah membentuk *even cycle* maka tidak ada prediksi yang diperlukan maka hanya perlu mendeteksi *odd cycle* saja pada algoritma ini.

Menurut Brian M. Scott pada situs *math.stackexchange.com*, Graf adalah bipartit jika dan hanya jika graf tersebut tidak memiliki *odd cycle* [10]. *Proof*, jika *G* adalah bipartit dengan *vertex U* dan *V*, setiap *edge* pasti dari *U* ke *V* dan sebaliknya. Untuk membentuk *cycle*, diperlukan *edge* dengan jumlah genap, sehingga graf tersebut tidak memiliki *odd cycle*.

Jika graf adalah bipartit dengan *sets U* dan *V*, maka tiap - tiap elemen pada *sets U* dan *V* harus saling terhubung untuk membentuk *even cycle*. Jadi walaupun *vertex* pada *sets U* dan *V* tidak terhubung hasil prediksi akan menghubungkan *vertex* tersebut sehingga tiap – tiap *vertex* pada *sets U* akan terhubung dengan tiap – tiap *vertex* pada *sets V*. Namun untuk mempermudah perhitungan anomali diperlukan modifikasi pada *weight* yaitu dengan menetapkan *weight* tidak pada *edge* namun pada *vertex* tersebut. Besar *weight* tersebut juga dimodifikasi yaitu dengan mengacu pada *weight* pada *root* graf, contoh modifikasi *weight* dapat dilihat pada Gambar 2.6.2.1.

Representasi graf yang telah dibentuk seperti Gambar 2.6.2 akan dirubah menjadi graf bipartit sekaligus menentukan apakah graf tersebut adalah graf bipartit atau bukan. Graf bipartit yang dibentuk akan memiliki dua *sets* yang memiliki sifat *disjoint sets* dalam artian tiap *sets* akan memiliki himpunan yang unik atau berbeda tidak ada yang sama. Tiap – tiap *vertex* akan berada *sets*, lalu tiap *vertex* yang berhubungan akan berada pada *sets* yang berbeda sehingga memenuhi kriteria *disjoint sets*.



weight	*Sets*		*weight*
vertex - root	*U*	*V*	*vertex + root*
0	30	-10	3 + 0 = 3
2 - 15 = -13	40	20	15 + 0 = 15

Gambar 2.6.2.1 Modifikasi *weight*

Untuk anomali yang disebut *weight* pada graf tersebut tidak akan sama seperti graf pada awalnya. Pada graf bipartit ini *weight* tersebut akan berpacu pada *root* atau *vertex* awal pada graf yang akan dilakukan *traversal* menggunakan *Breadth First Search* (BFS) seperti yang telah dijelaskan pada subbab 2.2 dan contoh pada Gambar 2.2.3. Dengan memberi acuan *root* sebagai *weight* pada graf bipartit akan memudahkan dan menambahkan kriteria baru untuk graf bipartit ini yaitu tiap *sets* yang berbeda dapat diprediksi anomalinya secara mudah. Contoh perubahan graf pada Gambar 2.6.1 menjadi graf bipartit serta perubahan *weight* nya dapat dilihat pada Gambar 2.6.2.2.

Root yang digunakan pada graf bipartit adalah *random* namun untuk contoh pada Gambar 2.6.2.2 adalah *vertex* dengan nilai 30 namun penulis menggunakan masukan awal pada soal untuk menjadi *root* pada graf tersebut untuk memudahkan implementasi program.

weight	*Sets*		*weight*
vertex - root	*U*	*V*	*vertex + root*
0	30		

weight	*Sets*		*weight*
vertex - root	*U*	*V*	*vertex + root*
0	30	-10	3
		20	15

weight	*Sets*		*weight*
vertex - root	*U*	*V*	*vertex + root*
0	30	-10	3
-13	40	20	15

Gambar 2.6.2.2 Perubahan Representasi Graf menjadi Graf Bipartit

Modifikasi *weight* untuk graf bipartit berpacu pada *root* dari graf tersebut, lalu dapat dilihat pada Gambar 2.6.2.2 bahwa *weight* pada *sets U* adalah *vertex – root* dan *weight* pada *sets V* adalah *vertex + root*. Seperti yang telah dijelaskan tadi bahwa modifikasi *weight* pada graf bipartit tersebut akan menambahkan kriteria baru yaitu tiap elemen pada *sets* yang berbeda memiliki anomali yang dapat diprediksi sebagai contoh graf pada Gambar 2.6.2 yang telah dirubah menjadi graf bipartit seperti pada Gambar 2.6.2.2 tidak diketahui berapa anomalinya pada koordinat (-10, 40) atau (40, -10) namun dengan hasil modifikasi tersebut kita dapat mudah mendapatkan berapa anomalinya hanya dengan menambahkan *weight* masing – masing *vertex* yaitu -13 + 3 = -10 maka anomali pada koordinat (-10, 40) atau (40, -10) adalah -10 perhitungan ini adalah benar sebagaimana dibandingkan dengan perhitungan sederhana seperti Gambar 2.5.4.

Untuk mendapatkan *weight* pada masing – masing *vertex* dapat dilakukan dengan cara berikut

1. *Weight* pada *root vertex* adalah 0
2. *Weight* pada masing – masing *sets* adalah *weight* dari *source* ke *neighbor vertex* dikurangi *source* ke *root vertex*, atau secara formal *w(s → n) – w(s → r)* dimana *w* adalah *weight*, *s* adalah *source*, *n* adalah *neighbor*, dan *r* adalah *root*.

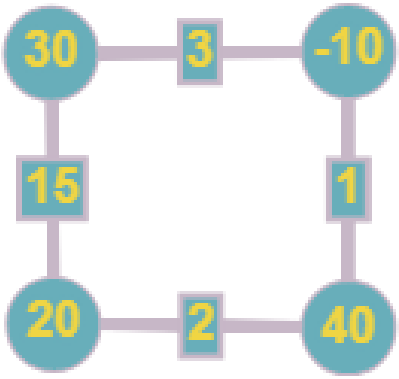
Perhitungan *weight* pada Gambar 2.6.2.2 dapat dilihat pada Gambar 2.6.2.3.

weight	*Sets*		*weight*
vertex - root	*U*	*V*	*vertex + root*
0	30		

weight	*Sets*		*weight*
vertex - root	*U*	*V*	*vertex + root*
0	30	-10	3 - 0 = 3
		20	15 - 0 = 15

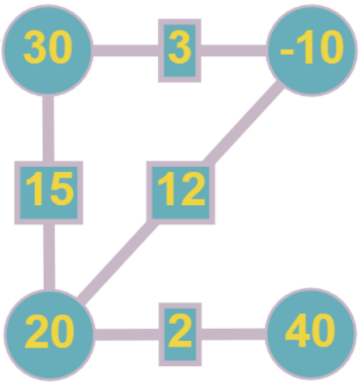
weight	*Sets*		*weight*
vertex - root	*U*	*V*	*vertex + root*
0	30	-10	3
2 - 15 = -13	40	20	15

Gambar 2.6.2.3 Perhitungan *Weight* Modifikasi pada Graf Bipartit



Gambar 2.6.2.4 Representasi Graf Hasil Prediksi 1

Hal yang perlu diperhatikan selanjutnya adalah ketika graf tersebut bukan lah graf bipartit yaitu jika ditemukan ada *vertex* yang saling berhubungan berada pada *sets* yang sama, maka semua generator yang terletak pada koordinat diagonal graf tersebut akan ditemukan nilai anomalinya. Jika dilihat pada Gambar 2.6.1.3 maka koordinat diagonal tersebut adalah (30, 30), (-10, -10), (40, 40), dan (20, 20).



Gambar 2.6.2.5 Contoh *Odd Cycle Graph*

Sebagai contoh kita tambahkan satu titik anomali pada koordinat (20, -10) atau (-10, 20) adalah 12 maka anomali yang ditemukan akan berada pada koordinat *root* graf tersebut yaitu koordinat (30, 30) contoh bukan graf bipartit atau graf yang memiliki *odd cycle* tersebut dapat dilihat pada Gambar 2.6.2.4

Jika ditemukan bahwa graf tersebut bukanlah graf bipartit yaitu dengan menentukan apakah graf tersebut memiliki siklus ganjil maka algoritma yang dilakukan adalah seperti berikut

1. Tandai bahwa graf tersebut bukanlah graf bipartit dan simpan nilai *root* nya
2. Lakukan *traversal* BFS dan jadikan graf bipartit hingga *traversal* selesai
3. Iterasi dari *root* hingga *vertex* akhir dan ubah *weight* tersebut menjadi *weight vertex*

Sebagai contoh dari hasil algoritma tersebut dapat dilihat pada Gambar 2.6.2.5.

weight	*Sets*		*weight*
vertex - root	*U*	*V*	*vertex + root*
0	30		

weight	*Sets*		*weight*
vertex - root	*U*	*V*	*vertex + root*
0	30	-10	3 - 0 = 3
		20	15 - 0 = 15

-10 dan 20 saling terhubung namun berada pada sets yang sama

root = (w(-10) + w(20) - w(-10, 20))/2
root = (3 + 15 - 12) / 2 = 3

weight	*Sets*		*weight*
vertex - root	*U*	*V*	*vertex + root*
0	30	-10	3
2 - 15 = -13	40	20	15

Ubah *weight* menjadi *weight vertex*

weight	*Sets*		*weight*
vertex - root	*U*	*V*	*vertex + root*
3	30		

Gambar 2.6.2.6 Perhitungan *Weight* Modifikasi pada *Odd Cycle Graph*

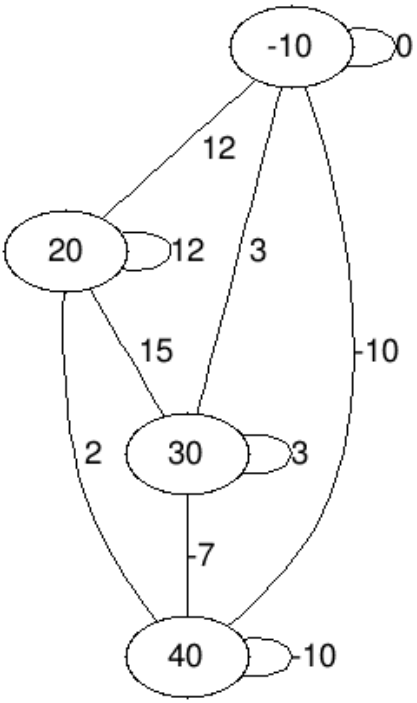
weight	*Sets*		*weight*
vertex - root	*U*	*V*	*vertex + root*
3	30	-10	0

weight	*Sets*		*weight*
vertex - root	*U*	*V*	*vertex + root*
3	30	-10	0
		20	12

weight	*Sets*		*weight*
vertex - root	*U*	*V*	*vertex + root*
3	30	-10	0
-10	40	20	12

Gambar 2.6.2.7 Perhitungan *Weight* Modifikasi pada *Odd Cycle Graph 2*

Sehingga hasil prediksi dari data pada Gambar 2.6.2.5 akan direpresentasikan sebagai graf seperti pada Gambar 2.6.2.6 dan Gambar 2.6.2.7. Namun pada permasalahan *Electrical Pollution*, graf yang dihasilkan bisa lebih atau sama dengan 1 maka akan ada lebih dari atau sama dengan 1 *root* yang akan diiterasi untuk perhitungan *weight* modifikasi



Gambar 2.6.2.8 Representasi Graf Hasil Prediksi 2

[*Halaman ini sengaja dikosongkan*]

BAB III
ANALISIS DAN PERANCANGAN

Pada bagian ini akan dijelaskan analisis dan desain sistem yang digunakan untuk menyelesaikan permasalahan pada Tugas Akhir ini.

3.1 Deskripsi Umum Sistem

Pada sistem ini terdapat 4 fungsi lain yaitu GetID yang akan dijelaskan pada bab 3.2, InsertGraph yang akan dijelaskan pada bab 3.3, Predict yang akan dijelaskan pada bab 3.4, dan GetAnomali yang akan dijelaskan pada bab 3.5. *Pseudocode* Fungsi Main ditunjukan pada Gambar 3.1.1.

```
1   m = Input() // Jumlah koordinat anomali yang diketahui
2   q = Input() // Jumlah koordinat anomali yang diprediksi
3   for m != 0 and q != 0
4       for 1 to m
5           x = GetID(Input())
6           y = GetID(Input())
7           a = Input()
8           G = InsertGraph(x, y, a)
9       P = Predict(G) //Prediksi anomali menggunakan Graf
10      for 1 to q
11          x = Input()
12          y = Input()
13          a = GetAnomali(P, x, y)
14      m=Input()
15      q=Input()
```

Gambar 3.1.1 *Pseudocode* Fungsi Main

Pertama sistem akan diberikan 2 masukan bilangan bulat yaitu *m* (jumlah koordinat anomali yang diketahui) dan *q* (jumlah koordinat anomali yang akan diprediksi). Untuk setiap *m* akan diberikan 3 masukan bilangan bulat yaitu *x* (koordinat X), *y*

(koordinat Y), *a* (anomali). Variabel *x* urutan atau ID *vertex* dari koordinat X begitu juga dengan variabel *y*. Lalu variabel *x, y* dan *a* akan dimasukan dan direpresentasikan dalam bentuk graf menggunakan fungsi InsertGraph.

Setelah koordinat anomali yang diketahui sejumlah *m* sudah direpresentasikan dalam bentuk graf maka dilakukin fungsi Predict untuk memprediksi koordinat anomali dengan informasi yang sudah ada.

Terakhir sistem akan mendapatkan anomali pada koordinat X dan Y lalu print hasil tersebut dengan fungsi GetAnomali.

3.2 Desain Fungsi GetID

Fungsi GetID digunakan untuk mendapatkan urutan atau ID dari *vertex*, selain itu fungsi GetID akan menghapus isi graf yang digunakan pada *test case* sebelumnya. *Pseudocode* Fungsi GetID ditunjukkan pada Gambar 3.2.1.

```
GetID(Hash, x, n, G)
1  if Hash[x] not null
2      return Hash[x]
3  else
4      clear G.edge[n]
5      Hash[x] = n++
6      return
G – graf,
Hash – array,
x – kunci index,
n – jumlah vertex
```

Gambar 3.2.1 *Pseudocode* Fungsi GetID

Fungsi ini digunakan karena batas dari koordinat X dan Y adalah $-10^4 <= X, Y <= 10^4$, sedangkan struktur data *array* pada C++ tidak dapat menggunakan indeks bilangan bulat negatif. Selain itu juga untuk memudahkan looping pada traversal dan meminimalisir *run time* sehingga sistem berjalan dengan efisien.

3.3 Desain Fungsi InsertGraph

Fungsi InsertGraph digunakan untuk merepresentasikan koordinat anomali menjadi graf terhubung, maupun tidak terhubung. Variabel *u* dan *v* adalah urutan atau ID dari koordinat *X* dan *Y*. *Pseudocode* Fungsi InsertGraph dapat dilihat pada Gambar 3.3.1

```
InsertGraph(e, u, v, w)
1  e[v] = pair(u, w)
2  e[u] = pair(v, w)
3  return

e – array of pair,
u – ID vertex1,
v – ID vertex2,
a – bobot
```

Gambar 3.3.1 *Pseudocode* Fungsi InsertGraph

3.4 Desain Fungsi Predict

Fungsi Predict digunakan untuk memprediksi anomali pada koordinat tertentu dari koordinat anomali yang sudah diketahui. Fungsi ini memanfaatkan traversal graf BFS, struktur data graf bipartit, dan algoritma deteksi siklus ganjil. *Pseudocode* Fungsi Predict dapa dilihat pada Gambar 3.4.1.

Traversal BFS sudah dimodifikasi karena graf yang dihasilkan dari fungsi sebelumnya terdiri dari beberapa graf tidak terhubung sehingga memiliki beberapa *root*, maka *flag* pada BFS tidak menggunakan *true* atau *false* melainkan menggunakan urutan *root*.

Struktur data graf bipartit menggunakan integer untuk menetapkan *set vertex* yaitu 1 dan -1, *vertex* pertama yang terpilih akan otomatis ditetapkan pada *set* 1. Tiap *vertex* yang berhubungan

```
Initialize col[0..G.countVertex] to 0
Initialize pos[0..G.countVertex] to 0
Predict(G, col, pos, a, b)
1   for i = 0..G.countVertex
2       if pos[i] != 0
3           continue
4       x = INFINITE of Integer
5       pos[i] = ++col
6       a[i] = 1
7       b[i] = 0
8       Q = empty queue
9       enqueue(Q, i)
10      while Q is not empty
11          u = dequeue(Q)
12          for each v adjacent to u
13              if pos[v] = 0
14                  pos[v] = col
15                  a[v] = - a[u]
16                  b[v] = u.weight – b[u]
17                  enqueue(Q, v)
18              else if a[u] != a[v]
19                  x = (weightOf(u, v) – b[u] – b[v]) /
                    (a[u] + a[v])
20      if x != INFINITE of Integer
21          for each q in Q
22              b[q] += a[q] * x
23              a[q] = 0

G – Graf
col – Jumlah root pada graf
pos – Tanda bahwa vertex telah dikunjungi
a – Set pada struktur data graf bipartit
b – Bobot pada graf bipartit
```

Gambar 3.4.1 *Pseudocode* Fungsi Predict

dengan *vertex* dan mempunyai *set* akan ditetapkan pada *set* sebaliknya (jika 1 maka *set vertex* adalah -1).

Algoritma untuk mendeteksi siklus ganjil mendeteksi ketika dua buah *vertex* berhubungan dan berada pada *set* yang sama maka dapat disimpulkan bahwa graf tersebut memiliki siklus ganjil.

3.5 Desain Fungsi GetAnomali

Fungsi GetAnomali digunakan untuk mendapatkan dan *print* hasil prediksi anomali pada koordinat tertentu. *Pseudocode* Fungsi GetAnomali dapat dilihat pada Gambar 3.5.1.

```
GetAnomali(u, v, Hash, flag)
1  if not Hash[u] exist and Hash[v] exist
2      print '*'
3  else
4      u = GetID(u)
5      v = GetID(v)
6      if pos[u] != pos[v]
7          if a[u] or a[v]
8              print '*'
9          else
10             print b[u] + b[v]
11     else
12         if a[u] = a[v]
13             print '*'
14         else if u != v
15             print b[u] + b[v]
16         else
17             print b[u]
18 print '\n'
x – koordinat X, y – koordinat Y
Hash – array
pos – array
```

Gambar 3.5.1 *Pseudocode* Fungsi GetAnomali

3.6 Desain Fungsi FastScan

Fungsi FastScan digunakan untuk menerima input dari user lebih cepat agar *run time* lebih efisien. Dengan memanfaatkan fungsi *getchar()* pada C++ maka input akan lebih efisien karena program langsung membaca ASCII tersebut. Namun ada cara yang lebih efisien yaitu menggunakan *getchar_unlocked()* selain langsung membaca ASCII input akan langsung dimasukan pada memori tanpa *binding*. *Pseudocode* Fungsi FastScan dapat dilihat pada Gambar 3.6.1.

```
FastScan(number)
1   negative = false
2   number = 0
3   c = getchar_unlocked()
4   if c == '-'
5       negative = true
6       c = getchar_unlocked()
7   for c > 47 && c < 58
8       number = number * 10 + c – 48
9   if not negative
10      number *= -1
```

Gambar 3.6.1 *Pseudocode* Fungsi FastScan

BAB IV
IMPLEMENTASI

Pada bab ini akan dijelaskan implementasi dari algoritma dan struktur data berdasarkan desain yang telah dilakukan.

4.1 Lingkungan Implementasi

Lingkungan implementasi menggunakan sebuah komputer dengan spesifikasi perangkat lunak dan perangkat keras seperti tercantum pada Tabel 4.1.

Tabel 4.1 Spesifikasi Lingkungan Implementasi

No.	Jenis Perangkat	Spesifikasi
1	Perangkat Keras	• *Processor* Intel Core i7-6700HQ CPU @ 2.60GHz • *Memory* 8GB 1333MHz DDR3
2	Perangkat Lunak	• Sistem operasi Windows 10 Pro • *Integrated Development Environment* Dev-C++ 5.3.0.3

4.2 Pendefinisian *Preprocessor Directives*

Pada bagian ini akan dijelaskan beberapa pendefinisian dari *preprocessor directives* yang akan digunakan dalam program ini. Penggunaan *preprocessor directive* ditujukan untuk mempermudah dan mempersingkat implementasi. Implementasi dari *preprocessor directive* ditunjukkan pada Kode Sumber 4.1 .

```
1  #define vii vector<pair<int, int> >
5  using namespace std;
```

Kode Sumber 4.1 Implementasi *Preprocessor Directive*

4.3 Implementasi Variabel Global

Pada bagian ini akan dijelaskan implementasi dari variabel global yang akan digunakan dalam program ini. Hal ini bertujuan untuk mempersingkat *run time* dari sistem karena selain lebih mudah untuk implementasi, *passing* variabel parameter dari fungsi main ke fungsi lain akan lebih lama karena C++ membutuhkan

waktu untuk meng *copy* variabel pada fungsi main ke variabel pada fungsi lain. Implementasi variabel global ditunjukan pada Kode Sumber 4.2.

```
1  const int maxn = 20001, INF = 0x3f3f3f3f;
2  map<int, int> Hash;
3  vii e[maxn];
4  int n, pos[maxn], a[maxn], b[maxn];
5  int que[maxn], L, R;
6  int m, q;
```

Kode Sumber 4.2 Implementasi Variabel Global

4.4 Implementasi Fungsi Main

Fungsi Main diimplementasikan sesuai *pseudocode* subbab 3.1. Pada awalnya sistem akan menetima masukan berupa banyaknya kasus uji, yaitu banyaknya koordinat anomali yang diketahui dan koordinat anomali yang akan diprediksi. Sistem akan berhenti ketika masukan koordinat anomali yang diketahui dan yang akan diprediksi adalah 0.

Selanjutnya, untuk setiap koordinat anomali yang diketahui fungsi Main akan menerima input *x, y,* dan *a* yang berarti koordinat *x y* memiliki anomali *a*, lalu fungsi main akan memanggil fungsi GetID dengan parameter *x* atau *y* bertujuan untuk mendapatkan urutan atau ID koordinat. Setelah itu fungsi main akan memanggil fungsi InsertGraph dengan parameter *x, y,* dan *a*. Setelah itu fungsi Main akan memanggil fungsi Predict untuk memprediksi anomali pada koordinat dari informasi yang telah diberikan.

Kemudian untuk setiap koordinat anomali yang akan diprediksi fungsi Main akan menerima input *x, y* yang berarti koordinat *x y* yang akan diprediksi. Fungsi Main akan memanggil fungsi GetAnomali dengan parameter *x, y*.Implementasi dari fungsi Main dapat dilihat pada Kode Sumber 4.3.

```
1  int main() {
2     FastScan(m);
3     FastScan(q);
```

```
4     while(m + q) {
5       n = 0;
6       map<int, int>().swap(Hash);
7       while(m--){
8         int u, v, w;
9         FastScan(u);
10        FastScan(v);
11        FastScan(w);
12        InsertGraph(u, v, w);
13      }
14      Predict();
15      while(q--){
16        int u, v;
17        FastScan(u);
18        FastScan(v);
19        GetAnomali(u, v);
20      }
21      printf("-\n");
22      FastScan(m);
23      FastScan(q);
24    }
25    return 0;
26  }
```

Kode Sumber 4.3 Implementasi Fungsi Main

4.5 Implementasi Fungsi GetID

Fungsi GetID berfungsi untuk mendapatkan urutan atau ID *vertex* dari koordinat anomali yang diberikan seperti yang dijelaskan pada bab 3.2. Implementasi dari Fungsi Main dapat dilihat pada Kode Sumber 4.4.

```
1   int GetID(int &x){
2     if(Hash.count(x))
3       return Hash[x];
4     vii().swap(e[n]);
5     return Hash[x] = n++;
6   }
```

Kode Sumber 4.4 Implementasi Fungsi GetID

vector<pair<int, int> >().swap(edge[countVertex]) adalah operasi untuk menghapus atau mereset *adjacency list* pada *edge* indeks ke *countVertex* yang sebelumnya digunakan untuk membuat graf pada *test case* sebelumnya.

4.6 Implementasi Fungsi InsertGraph

Fungsi InsertGraph berfungsi untuk membuat graf dari input koordinat anomali seperti yang dijelaskan pada bab 3.3. Implementasi dari fungsi InsertGraph dapat dilihat pada Kode Sumber 4.5.

```
1  void InsertGraph(int &u, int &v, int &w){
2     u = GetID(u);
3     v = GetID(v);
4     if(u == v)
5        w <<= 1;
6     e[u].push_back(make_pair(v, w));
7     e[v].push_back(make_pair(u, w));
8  }
```

Kode Sumber 4.5 Implementasi Fungsi InsertGraph

Pada implementasi dari fungsi InsertGraph adalah ketika u dan v sama maka w pada koordinat u, v adalah w /= 2. Karena informasi yang diberikan adalah jumlah anomali dari koordinat x (*u*) ditambah anomali dari koordinat y (*v*).

4.7 Implementasi Fungsi Predict

Fungsi Predict berfungsi untuk memprediksi anomali pada koordinat tertentu dari informasi yang sudah diberikan. Seperti yang sudah dijelaskan pada bab 3.4, Implementasi dari fungsi Predict dapat dilihat pada Kode Sumber 4.6.

```
1  void Predict() {
2     memset(pos, 0, n*sizeof(int));
3     for(int i=0, col=0; i<n; i++){
4        if(pos[i]) continue;
5        ++col;
```

```
6       int x = INF;
7       L = R = 0;
8       pos [i] = col;
9       a[i] = 1;
10      b[i] = 0;
11      que[R++] = i;
12      while(L < R){
13         int u = que[L++];
14         for(vii::iterator  it=e[u].begin();
                          it!=e[u].end(); ++it){
15            int v=it->first, w=it->second;
16            if(!pos[v]){
17               pos[v] = col;
18               a[v] = -a[u];
19               b[v] = w - b[u];
20               que[R++] = v;
21            }
22            else if(!(a[u] + a[v] == 0)){
23               x =  (w - b[u] - b[v])/
                        (a[u] + a[v]);
24         }
25      }
26      if(x == INF
27         for(int j=0; j<R; j++)
28            a[que[j]] = x;
29      else if(x != INF)
30         for(int j=0; j<R; j++){
31               b[que[j]] += a[que[j]] * x;
32               a[que[j]] = 0;
33         }
34   }
35 }
```

Kode Sumber 4.6 Implementasi Fungsi Predict

4.8 Implementasi Fungsi GetAnomali

Fungsi GetAnomali berfungsi untuk mendapatkan anomali pada suatu koordinat seperti yang sudah dijelaskan pada bab 3.5. Implementasi GetAnomali dapat dilihat pada Kode Sumber 4.7

```
1   void GetAnomali(int &u, int &v){
2      if(!(Hash.count(u) && Hash.count(v)))
3         putchar('*');
4      else{
5         u = GetID(u);
6         v = GetID(v);
7         if(pos[u] != pos[v]){
8            if(a[u] || a[v]) putchar('*');
           else printf("%d", b[u] + b[v]);
9      }
10     else{
11        if(a[u] + a[v] != 0)
12           putchar('*');
13        else if(u != v)
14           printf("%d", b[u] + b[v]);
15        else
16           printf("%d", b[u]);
17     }
18  }
19  putchar('\n');
```

Kode Sumber 4.7 Implementasi Fungsi GetAnomali

4.9 Implementasi Fungsi FastScan

Fungsi FastScan berfungsi untuk menerima input lebih cepat dan efisien seperti yang sudah dijelaskan pada bab 3.6. Implementasi fungsi FastScan dapat dilihat pada Kode Sumber 4.8

```
1   void FastScan(int &number) {
2      bool negative = false;
3      register int c;
4      number = 0;
```

```
5     c = getchar();
6     if (c=='-') {
7        negative = true;
8        c = getchar();
9     }
10    for (; (c>47 && c<58); c=getchar())
11       number = number *10 + c - 48;
12    if (negative)
13       number *= -1;}
14 }
```

Kode Sumber 4.8 Implementasi Fungsi FastScan

[*Halaman ini sengaja dikosongkan*]

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dijelaskan tentang uji coba dan evaluasi dari implementasi sistem yang telah dilakukan pada bab 4.

5.1 Lingkungan Uji Coba

Lingkungan uji coba menggunakan sebuah komputer dengan spesifikasi perangkat lunak dan perangkat keras seperti tercantum pada Tabel 5.1.

Tabel 5.1 Spesifikasi Lingkungan Uji Coba

No.	Jenis Perangkat	Spesifikasi
1	Perangkat Keras	• *Processor* Intel Core i7-6700HQ CPU @ 2.60GHz • *Memory* 8GB 1333MHz DDR3
2	Perangkat Lunak	• Sistem operasi Windows 10 Pro • *Integrated Development Environment* Dev-C++ 5.3.0.3

5.2 Skenario Uji Coba

Pada subbab ini akan dijelaskan skenario uji coba yang dilakukan. Skenario uji coba terdiri dari uji coba kebenaran dan uji coba kinerja. Uji coba kebenaran dilakukan dengan strategi penyelesaian yang telah dijelaskan pada subbab 2.4. Uji coba kebenaran akan menggunakan kasus berikut ini:

```
6 8
0 1 11
0 3 8
1 0 11
3 0 8
4 4 0
3 5 6
1 5
```

```
0 3
3 0
4 3
0 2
2 4
4 4
5 5
0 0
```

Gambar 5.2.1 Contoh Kasus Uji Coba

Uji coba kinerja dilakukan dengan membuat komparasi kinerja untuk melihat pengaruh batasan nilai variabel m dan q terhadap pertumbuhan waktu dari strategi penyelesaian yang telah dijelaskan pada subbab 2.4.

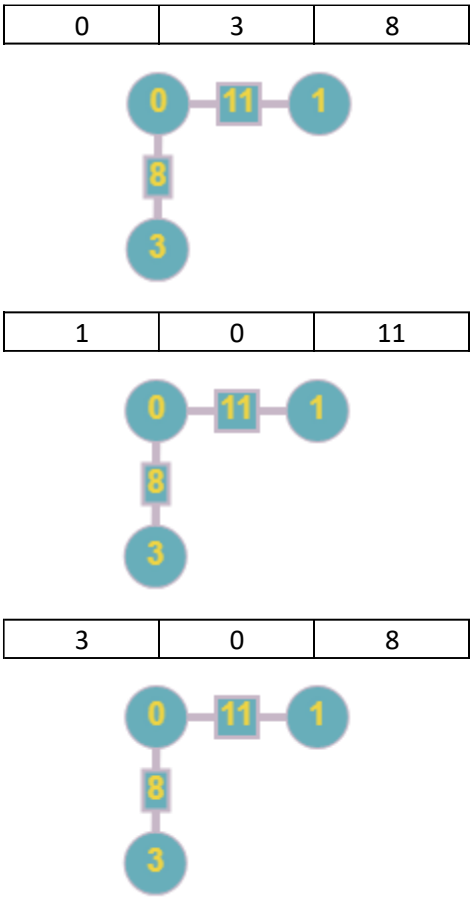
5.2.1 Uji Coba Kebenaran Naif

Uji coba kebenaran yang dilakukan adalah menganalisis kasus uji coba sederhana dengan langkah – langkah yang telah dijelaskan pada subbab 2.6 untuk memvalidasi kebenaran hasil keluaran program sesuai dengan hasil keluaran yang diharapkan, dan mengirimkan kode sumber ke URI Online Judge. Permasalahan yang diselesaikan adalah Electrical Pollution dengan kode soal 1334.

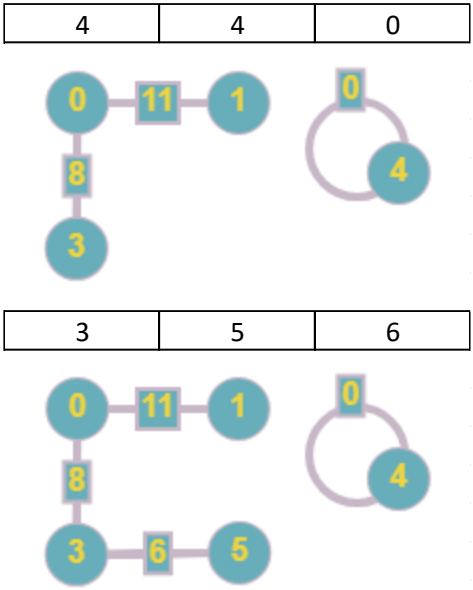
Contoh kasus uji coba dapat dilihat pada Gambar 5.2.1, pada tiap baris masukan akan dilakukan pembentukan graf seperti pada langkah berikut

0	1	11

Gambar 5.2.1.1 Pembentukan Graf (1)

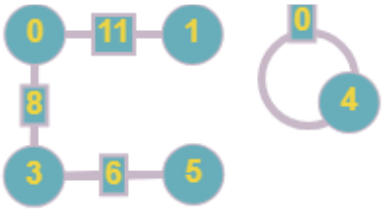


Gambar 5.2.1.2 Pembentukan Graf (2)



Gambar 5.2.1.3 Pembentukan Graf (3)

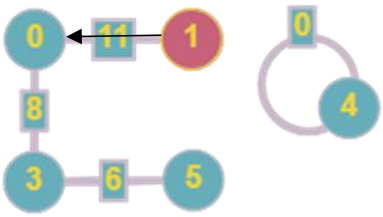
Setelah itu akan dilakukan prediksi terhadap masukan yang diberikan yaitu titik (1, 5). Langkah pertama yang dilakukan adalah mencari *path* antara 2 titik tersebut seperti langkah berikut



queue	1

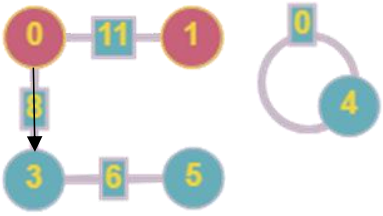
vertex	*source*
1	-

Gambar 5.2.1.4 Pencarian *Path* 1 ke 5 (1)



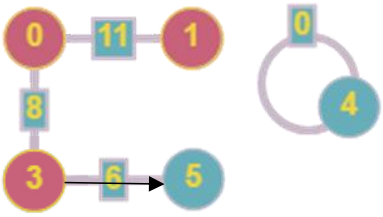
queue	0

vertex	*source*
1	-
0	1



queue	3

vertex	*source*
1	-
0	1
3	0



queue	-

vertex	*source*
1	-
0	1
3	0
5	3

Gambar 5.2.1.5 Pencarian *Path* 1 ke 5 (2)

Dari langkah pencarian path tersebut maka didapatkan *path* dengan rute 5 – 3 – 0 – 1 lalu akan diprediksi anomali-nya dengan melakukan persamaan seperti berikut

$w(1, 5) = w(5, 3) – w(3, 0) + w(0, 1)$
$w(1, 5) = 6 – 8 + 11 = 9$

Maka anomali pada titik(1, 5) adalah 9

Selanjutnya adalah pencarian pada titik (0, 3) dan titik (3, 0) dimana anomali pada titik tersebut sudah diketahui yaitu 8, sedangkan pencarian selanjutnya yaitu pada titik (4, 3) diketahui bahwa tidak ada *path* dari *vertex* 4 menuju *vertex* 3 sehingga titik tersebut tidak dapat diprediksi anomalinya dan menampilkan '*' ke *console*.

Masukan selanjutnya adalah pencarian pada titik (0, 2) lalu titik (2, 4) diketahui bahwa pada graf tersebut tidak memiliki *vertex* 2 sehingga titik tersebut tidak dapat diprediksi anomalinya dan menampilkan '*' ke *console*.

Setelah itu masukan selanjutnya adalah pencarian pada titik (4, 4) dimana anomali pada titik tersebut sudah diketahui yaitu 0, lalu pencarian selanjutnya pada titik (5, 5) yang belum diketahui titik anomalinya, lalu karena $x = y$ maka sistem tidak akan melakukan pencarian *path*.

Dari hasil analisis diatas menghasilkan keluaran yang sesuai. Setelah itu dilakukan uji kebenaran dengan mengumpulkan berkas kode sumber ke situs *URI Online Judge* ditunjukan pada Gambar 5.2.1.6

Gambar 5.2.1.6 Hasil Uji Coba Kebenaran pada Situs *URI Online Judge*

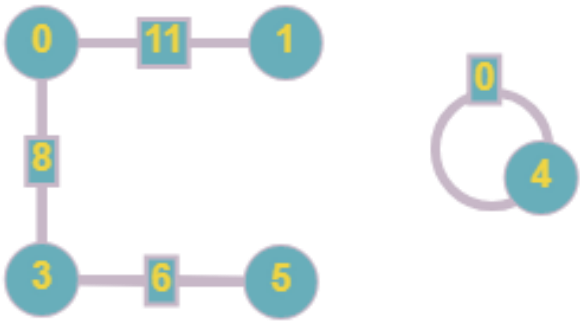
Diketahui bahwa hasil uji coba mendapatkan umpan balik *Time limi exceeded*. Waktu yang diperlukan untuk menyelesaikan permasalahan menggunakan lebih dari *limit* yaitu 3 detik sehingga penyelesaian naif dapat digunakan namun tidak optimal untuk menyelesaikan permasalahan pada situ *URI Online Judge Electrical Pollution* karna *time limit* pada persoalan tersebut adalah 3 detik

.

5.2.2 Uji Coba Kebenaran Algoritma *Cycle Finding*

Contoh kasus uji coba dapat dilihat pada Gambar 5.2.1, Graf yang terbentuk dari masukan tersebut terdapat pada Gambar 5.2.2.1.

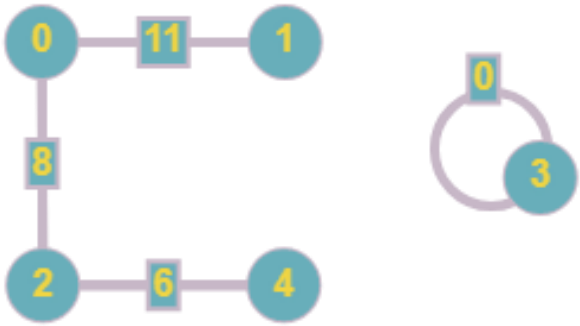
Langkah selanjutnya adalah mengubah graf tersebut menjadi graf bipartit. Pertama – tama ubah nilai *vertex* menjadi urutan atau ID dengan menggunakan fungsi GetID untuk mempermudah mengubah graf tersebut menjadi graf bipartit seperti pada contoh Gambar 5.2.2.2.



Gambar 5.2.2.1 Contoh Graf pada Kasus Uji

GetID(0) → 0
GetID(1) → 1
GetID(3) → 2
GetID(4) → 3
GetID(5) → 4

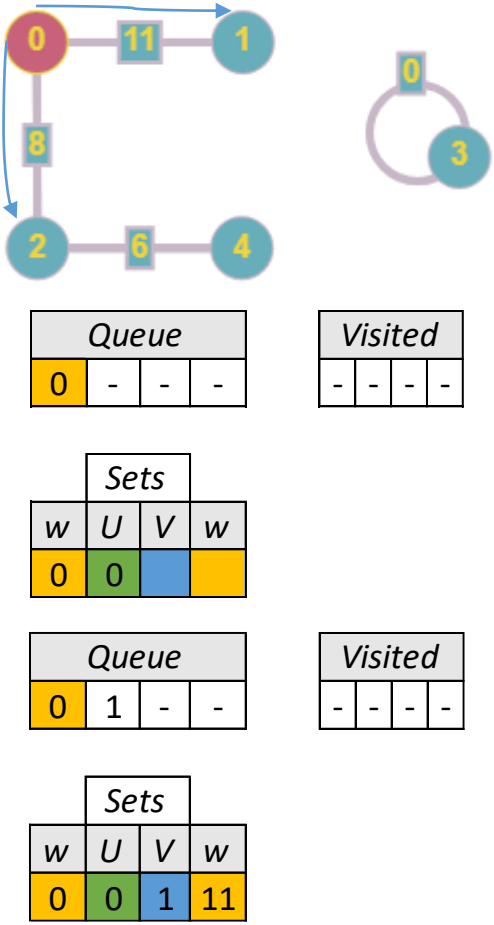
Gambar 5.2.2.2 Contoh Hasil GetID pada Kasus Uji Coba



Gambar 5.2.2.3 Contoh Graf setelah Fungsi GetID

Dari hasil fungsi GetID tersebut representasi graf-nya adalah seperti pada contoh Gambar 5.2.2.3. Hasil dari graf dengan *vertex* yang berurutan atau memiliki ID tersebut kita akan memprediksi nilai anomali pada koordinat yang lain. Untuk merubah graf menjadi graf bipartit diperlukan *traversal* graf. Penulis menggunakan *traversal* BFS karena diperlukan untuk menemukan atau *traverse* dari tetangga ke tetangga.

Dengan menggunakan fungsi Predict sistem akan melakukan *traversal* dari tiap – tiap graf yang ada dimulai dari *vertex* pertama yaitu *vertex* dengan id 0 dengan menggunakan BFS hingga vertex terakhir. Contoh merubah graf pada Gambar 5.2.2.3 menjadi graf bipartit sebagai berikut.

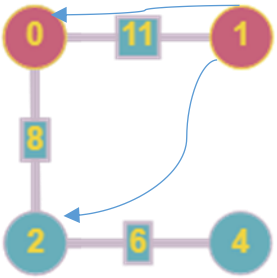


Gambar 5.2.2.4 Ilustrasi Fungsi Predict 1

Queue			
0	1	2	-

Visited			
-	-	-	-

	Sets		
w	U	V	w
0	0	1	11
		2	8

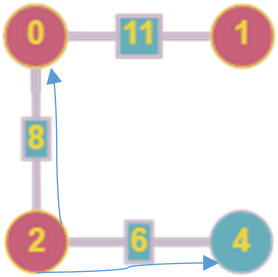


Queue			
0	1	2	-

Visited			
0	-	-	-

	Sets		
w	U	V	w
0	0	1	11
		2	8

Gambar 5.2.2.5 Ilustrasi Fungsi Predict 2



Queue			
0	1	2	-

Visited			
0	1	-	-

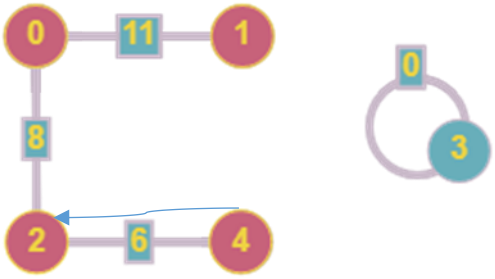
	Sets		
w	*U*	*V*	*w*
0	0	1	11
		2	8

Queue			
0	1	2	4

Visited			
0	1	-	-

	Sets		
w	*U*	*V*	*w*
0	0	1	11
-2	4	2	8

Gambar 5.2.2.6 Ilustrasi Fungsi Predict 3



Queue			
0	1	2	4

Visited			
0	1	2	-

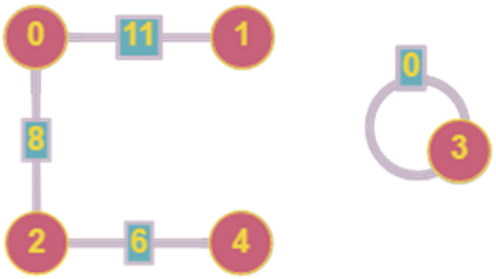
	Sets		
w	U	V	w
0	0	1	11
-2	4	2	8

Queue			
0	1	2	4

Visited			
0	1	2	4

	Sets		
w	U	V	w
0	0	1	11
-2	4	2	8

Gambar 5.2.2.7 Ilustrasi Fungsi Predict 4



Queue
3

Visited
-

	Sets		
w	*U*	*V*	*w*
0	3		

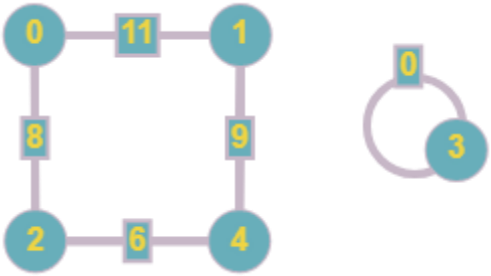
Queue
3

Visited
3

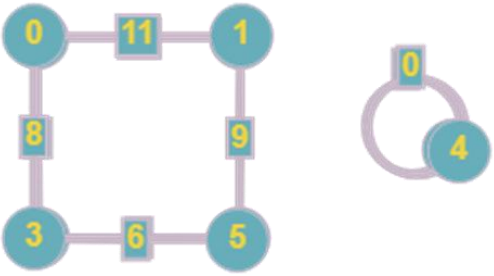
	Sets		
w	*U*	*V*	*w*
0	3		

Gambar 5.2.2.8 Ilustrasi Fungsi Predict 5

Dari graf bipartit yang dihasilkan oleh fungsi predict, dihasilkan dua graf dengan *root* 0 dan *root* 3. Dengan mengikuti kriteria dari *weight* modifikasi dimana graf bipartit yang dihasilkan bahwa elemen graf bipartit pada *sets* yang berbeda akan selalu dapat diprediksi dengan cara sederhana yaitu dijumlahkan, maka dari graf tersebut anomali yang terprediksi terdapat pada *vertex* 1 dan *vertex* 4 yaitu dengan anomali sebesar 9. Representasi graf yang dihasilkan oleh hasil prediksi adalah seperti berikut



Gambar 5.2.2.9 Contoh Representasi Graf Hasil Prediksi dengan ID



Gambar 5.2.2.10 Contoh Representasi Graf Hasil Prediksi

Hasil dari fungsi Predict tersebut sudah selesai, lalu sistem akan menerima input untuk menampilkan apakah data yang sudah diprediksi mampu meberikan besar anomali pada koordinat yang diberikan. Jika sistem mampu memprediksi anomali tersebut maka sistem akan memberikan keluaran berupa besar anomali tersebut, jika tidak maka sistem akan memberikan keluaran berupa karakter '*'. Masukan untuk tanda bahwa sistem telah selesai digunakan atau *End Of File* adalah masukan berupa angka 0 dan 0, lalu pada kasus uji coba, koordinat yang akan diprediksi anomalinya adalah sebagai berikut.

```
1 5
0 3
3 0
4 3
0 2
2 4
4 4
5 5
0 0
```

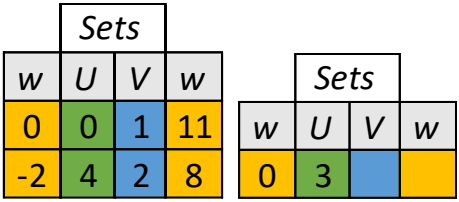
Gambar 5.2.2.11 Masukan Dari Uji Kasus untuk Prediksi Anomali

Dari masukan tersebut sistem akan menggunakan fungsi GetAnomali untuk menemukan hasil prediksi yang telah dilakukan. Fungsi Parameter yang diperlukan untuk fungsi tersebut adalah koordinat *x* dan *y* titik anomali yang akan didapatkan anomalinya. Fungsi GetAnomali menggunakan graf bipartit dengan modifikasi *weight* yang telah dibuat, langkah fungsi GetAnomali adalah sebagai berikut.

1 4
0 2
2 0
3 2
0 x
x 3
3 3
4 4
0 0 → Keluar program

Gambar 5.2.2.12 ID Masukan Dari Uji Kasus Untuk Prediksi Anomali

Sistem akan mendapatkan anomali yang telah diprediksi dengan menggunakan graf bipartit yang telah dibuat yaitu sebagai berikut.



Gambar 5.2.2.13 Graf Bipartit untuk Fungsi GetAnomali

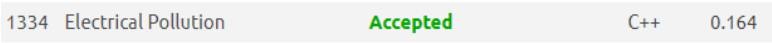
Pertama fungsi GetAnomali akan mengecak apakah masukan yang berupa koordinat dan menjadi *vertex* pada graf berada pada graf yang sama sebagai contoh (1, 4) *vertex* tersebut terdapat pada 1 graf. Jika *vertex* tersebut tidak berada pada graf yang sama maka sistem akan mengecek apakah *vertex* tersebut berada pada graf bukan bipartit, jika iya maka hanya perlu menambahkan *weight* modifikasi untuk mendapatkan prediksinya jika tidak maka data tidak cukup untuk melakukan prediksi dan keluarkan karakter '*'.

Vertex tersebut berada pada graf yang sama maka sistem akan mengecek kembali apakah graf tersebut merupakan graf bipartit atau bukan. Jika graf tersebut bipartit maka hasil prediksi akan didapatkan, lalu jika tidak sistem akan mengecek kembali apakah *vertex* tersebut berada pada *sets* yang berbeda. Jika tidak maka data yang diberikan tidak cukup untuk sistem memprediksi anomali pada koordinat tersebut dan jika *vertex* tersebut berada pada *sets* yang berbeda maka sistem mampu untuk memprediksi anomali pada koordinat tersebut. Contoh ilustrasi fungsi GetAnomali adalah sebagai berikut.

1 4 → satu graf → *sets* berbeda → Anomali = 9
0 2 → satu graf → *sets* berbeda → Anomali = 8
2 0 → satu graf → *sets* berbeda → Anomali = 8
3 2 → graf berbeda → graf bipartit → Print '*'
0 x → ID *vertex* tidak ada → Print '*'
x 3 → ID *vertex* tidak ada → Print '*'
3 3 → satu graf → graf bukan bipartit → Anomali = 0
4 4 → satu graf → graf bipartit → Print '*'
0 0 → Keluar program

Gambar 5.2.2.14 Ilustrasi Fungsi GetAnomali

Dari hasil analisis diatas menghasilkan keluaran yang sesuai. Setelah itu dilakukan uji kebenaran dengan mengumpulkan berkas kode sumber ke situs *URI Online Judge* ditunjukan pada Gambar 5.2.2.15



Gambar 5.2.2.15 Hasil uji coba kebenaran pada situs *URI Online Judge*

Hasil uji coba yang telah dilakukan program mendapat umpan balik *Accepted*. Waktu yang dibutuhkan program untuk menyelesaikan permasalahan tersebut adalah 0.164 detik.

5.2.3 Uji Coba Kinerja

Uji Coba pada penyelesaian naif telah didapatkan umpan balik *Time limit exceeded* dari *URI Online Judge*, dapat diketahui bahwa langkah utama pada penyelesaian ini adalah

1. Merubah masukan menjadi graf dengan melakukan deteksi ketika terdapat masukan *odd cycle* maka dapatkan *path* antara *x* dan *y* sehingga memiliki kompleksitas *Breadth First Search* yaitu *O(V+E)* dengan *V* adalah jumlah *vertex* dan *E* adalah jumlah *edge*
2. Ketika prediksi, *worst case* untuk kompleksitas permasalahan adalah melakukan pencarian *path* yaitu *O(V+E)*
3. Sehingga dapat disimpulkan *worst case* pada penyelesaian naif tiap *M* adalah *O(V+E)* dan tiap *Q* adalah *O(V+E)*

Uji Coba pada penyelesaian menggunakan algoritma *cycle finding* telah berhasil mendapatkan umpan balik *Accepted* dengan urutan pertama pada *URI Online Judge* seperti pada Gambar 5.2.3.1 yang *valid* per tanggal 29 November 2018.

RANKING	SUBMISSION	USER	LANGUAGE	RUNTIME
1	12389648	Firza Gustama	C++	0.164
2	8747557	pabloskimg	C++	0.248
3	2809529	Ricardo Martins	C++	0.260
4	5520119	Maycon Alves	C++	0.260
5	8487678	Luis Fernando Veronese Trivelatto	C++	0.264
6	7222577	Trupe da Biologia	C++	0.268
7	10127851	Rachmadi Rizki	C++	0.268
8	8717958	Paulo Cezar	C++	0.272
9	123338	Fernando Fonseca	C++	0.276
10	10953637	Lucas Soares	C++	0.280

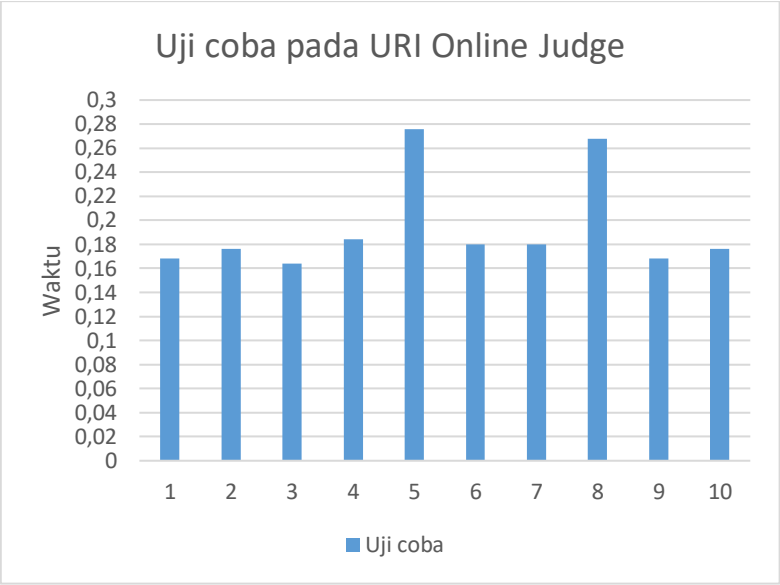
Gambar 5.2.3.1 Urutan pada *URI Online Judge*

Seperti yang telah dijelaskan pada bab 2 bahwa inti pada algoritma *cycle finding* untuk penyelesaian ini adalah

1. Mengubah graf menjadi graf bipartit menggunakan *Breadth First Search* yang memiliki kompleksitas *O(V+E)*
2. Ketika merubah menjadi graf bipartit jika didapatkan bahwa suatu *vertex* saling berhubungan pada *sets* yang sama akan dilakukan perhitungan yang memiliki kompleksitas *O(V)*
3. Sehingga kompleksitas *worst case* keseluruhan pada algoritma ini adalah *O(V+E) + O(V) = O(2V+E) = O(V+E)*

Maka perbedaan penyelesaian naif dan penyelesaian menggunakan algoritma adalah penyelesaian naif memiliki kompleksitas *O(V+E)* yang dilakukan sejumlah *M+Q* namun pada algoritma *cycle finding* kompleksitas-nya adalah *O(V+E*) namun hanya dilakukan sekali untuk menghitung semuanya sehingga didapatkan selisih waktu yang significant seperti yang sudah di uji kebenaranya dan kinerjanya pada *URI Online Judge*.

Lalu dengan penyelesaian menggunakan algoritma *cycle finding* ditunjukan bahwa grafik hasil uji coba pengumpulan sebanyak 10 kali ditunjukan pada Gambar 5.2.3.2. Dari hasil pengumpulan sebanyak 10 kali didapat waktu rata – rata program yaitu 0.194 detik.



Gambar 5.2.3.2 Hasil uji coba kinerja

BAB VI
KESIMPULAN

Pada bab ini akan dijelaskan kesimpulan dari hasil uji coba yang telah dilakukan.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap implementasi solusi untuk permasalahan *Electrical Pollution* dalam mencari sebuah bilangan dari rentang yang telah ditentukan dengan pertanyaan seminimum mungkin, dapat diambil kesimpulan sebagai berikut:

1. Permasalahan mencari *odd cycle* pada graf bipartit dari rentang yang telah diberikan telah berhasil diselesaikan dengan teori-teori dan *lemma* yang telah dicantumkan pada bab 2
2. Penggunaan *Map* untuk menyimpan data dengan *index* dibawah 0 dan fungsi GetID dapat dengan menyelesaikan permasalahan dengan efisien
3. Menggunakan kriteria graf bipartit yaitu graf yang tidak memiliki *odd cycle*, untuk menemukan *odd cycle* pada suatu graf dapat menyelesaikan permasalahan dengan efisien.
4. Waktu dan penggunaan memori rata-rata yang diperlukan untuk menyelesaikan permasalahan *Electrical Pollution* dari 10 kali uji coba pada daring *URI Online Judge* adalah 0.194 detik
5. Rata-rata waktu yang diperlukan untuk menyelesaikan kasus uji pada daring *URI Online Judge* merupakan yang terbaik dibandingkan dengan implementasi naif

6.2 Saran

Saran yang diberikan adalah penggunaan deteksi *odd cycle* dengan cara merubah graf menjadi graf bipartit menggunakan *traversal Breadth First Search*, dapat dirubah menjadi deteksi *odd cycle* tanpa merubah graf sehingga masukan akan langsung

menjadi graf bipartit tanpa melalui pembentukan graf terlebih dahulu.

DAFTAR PUSTAKA

[1] T. Harju, "Graph Theory," Finland, Departement of Mathematics University of Turku, 2012.

[2] L. A, The Design & Analysis of Algorithms, Pearson Education, Inc., 2012.

[3] N. Nisse, "Graph Theory and Optimization Weighted Graphs Shortest Paths & Spanning Trees," Université Côte d'Azur, Inria, France, 2018.

[4] E. Weisstein, "A Wolfram Web Resource," Math World, [Online]. Available: http://mathworld.wolfram.com/CycleGraph.html. [Accessed 10 Desember 2018].

[5] S. Jain, V. Bajpai, S. Goel, D. Singh and S. Baranwal, "GeeksforGeeks," [Online]. Available: https://www.geeksforgeeks.org/set-in-cpp-stl/. [Accessed 10 January 2019].

[6] B. R. Arunkumar and R. Komala, "Applications of Bipartite Graph in diverse fields including cloud computing," *International Journal Of Modern Engineering Research (IJMER)*, vol. V, no. 7, p. 1, 2015.

[7] P. Garg, "HackerEarth," [Online]. Available: https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/. [Accessed 10 Desember 2018].

[8] "CPP Reference," [Online]. Available: http://en.cppreference.com/w/cpp/utility/pair. [Accessed 10 Desember 2018].

[9] "Map in C++ Standard Template Library (STL)," GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/map-associative-containers-the-c-standard-template-library-stl/. [Accessed 12 Desember 2018].

[10] B. M. Scott, "Proof a graph is bipartite if and only if it contains no odd cycles," Mathematics Stack Exchange, [Online]. Available: https://math.stackexchange.com/questions/311665/proof-a-graph-is-bipartite-if-and-only-if-it-contains-no-odd-cycles. [Accessed 14 Januari 2019].

LAMPIRAN A
HASIL UJI COBA PADA URI ONLINE JUDGE SEBANYAK 10 KALI

Berikut merupakan lampiran hasil uji coba pengumpulan berkas kode solusi sebanyak 10 kali dari grafik yang ditampilkan pada Gambar 5.2.3.2

#		PROBLEM	ANSWER	LANGUAGE	TIME
12660163	1334	Electrical Pollution	**Accepted**	C++	0.176
12660161	1334	Electrical Pollution	**Accepted**	C++	0.168
12660160	1334	Electrical Pollution	**Accepted**	C++	0.268
12660156	1334	Electrical Pollution	**Accepted**	C++	0.180
12660155	1334	Electrical Pollution	**Accepted**	C++	0.180
12660154	1334	Electrical Pollution	**Accepted**	C++	0.276
12660151	1334	Electrical Pollution	**Accepted**	C++	0.184
12660149	1334	Electrical Pollution	**Accepted**	C++	0.164
12660147	1334	Electrical Pollution	**Accepted**	C++	0.176
12660139	1334	Electrical Pollution	**Accepted**	C++	0.168

Gambar A.1 Hasil Pengumpulan Kode Sumber Sebanyak 10 Kali

[*Halaman ini sengaja dikosongkan*]

BIODATA PENULIS

Muhammad Firza Gustama, lahir di Surabaya tanggal 12 Agustus 1997. Penulis merupakan anak kedua dari 3 bersaudara. Penulis telah menempuh pendidikan formal TK Dharma Wanita, SD Tawangsari Permai III Sidoarjo (2003-2009), SMPN 6 Surabaya (2009-2012) , SMA Muhammadiyah 2 Surabaya (2012-2015). Penulis melanjutkan studi kuliah program sarjana di Departemen Teknik Informatika ITS.

Selama kuliah di Teknik Informatika ITS, penulis mengambil bidang minat Algoritma Pemrograman (AP). Selama menempuh perkuliahan, penulis juga aktif mengikuti organisasi kemahasiswaan sebagai staff Hubungan Luar di Himpunan Mahasiswa Teknik Computer-Informatika (HMTC) ITS. Penulis juga aktif dalam kegiatan kepanitiaan Schematics sebagai staff Hubungan Masyarakat pada tahun 2016 dan menjadi staff ahli REEVA pada tahun 2017. Selain itu penulis pernah menjadi ketua di salah satu komunitas musik yang ada di ITS yaitu ITS Jazz pada tahun 2017 – 2018. Penulis dapat dihubungi melalui surel di muhfir_za@yahoo.com.